Theoretical Computer Science 114 (1993) 317–330
Elsevier

Note

A simple sequent calculus for partial functions

Morten Elvang-Gøransson

Centre of Cognitive Informatics, Roskilde University Center, P.O. Box 260, DK-4000 Roskilde, Denmark

Olaf Owe

Department of Informatics, University of Oslo, P.O. Box 1080, N-0316 Oslo, Norway

Communicated by M. Nivat
Received February 1992
Revised August 1992

Abstract

Elvang-Gøransson, M. and O. Owe, A simple sequent calculus for partial functions, Theoretical Computer Science 114 (1993) 317–330.

Usually, the extension of classical logic to a three-valued logic results in a complicated calculus, with side-conditions on the rules of logic in order to ensure consistency. One reason for the necessity of side-conditions is the presence of nonmonotonic operators. Another reason is the choice of consequence relation. Side-conditions severely violate the symmetry of the logic. By limiting the extension to monotonic cases and by choosing an appropriate consequence relation, a simple calculus for three-valued logic arises. The logic has strong correspondences to ordinary classical logic and, in particular, the symmetry of the Genzen sequent calculus (LK) is preserved, leading to a simple proof for cut elimination.

1. Introduction

Over the past years, numerous logics for handling partial functions and a "gap" in the truth values have been defined. Most of these so-called three-valued logics are obtained by extending the interpretation of the usual classical connectives, for instance, as suggested by Kleene, and by adding new connectives to reflect the increased

Correspondence to: O. Owe, Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway. Email addresses of the authors: elvang@cog.ruc.dk and olaf@ifi.uio.no.

expressive power needed to capture well-definedness properties, and then define a calculus which is complete with respect to such interpretations. In computer science applications, gaps may exist not only for truth values but also for all types involved, corresponding to "undefined" values. Thus, the term "three-valued" is not quite appropriate. We are interested primarily in formalisms where "gaps" correspond to meaningless values (or rather nonexisting values), for instance, caused by run-time errors or nontermination.

In this paper we question whether all these efforts have resulted in the "right" three-valued logic. Our response will be a three-valued calculus which has never seriously been considered before. The logic arises as a natural extension of the two-valued classical case to the three-valued monotonic case. As we shall see, its validity concept – together with an inductive definition of definedness – gives rise to a very simple proof system.

Section 2 justifies our interest in the calculus, which then is formally introduced through Sections 3 and 4. A few interesting properties of the calculus are described in Section 5 and some final remarks are placed in Section 6.

We assume that the reader is familiar with standard definitions from logic and denotational semantics. The applicability for three-valued logics is well motivated by others (refer e.g. [5]). We refer readers looking for a survey of three-valued logics to [1, 3, 5, 9, 14, 16].

2. Motivation and background

In order to formalize the use of assumptions (hypotheses) in a convenient way, we consider sequents of the form $A \vdash B$, expressing that if the *hypothesis* A is true, then we can entail that the *consequent* B is true as well. For now, we assume that A and B are single formulas (taking true as the default hypothesis). Thus, one entails true consequents from true hypotheses; however, in the classical setting we could as well have read that from nonfalse hypotheses we entail true consequents. This distinction is without importance in classical logic, but in logics with a third value this distinction is important, and we see it as the philosophical motivation for the logic discussed in this paper.

In a three-valued setting, there are four obvious ways of reading $A \vdash B$. Apart from the two possibilities mentioned above, one may say that nonfalse consequents are entailed from nonfalse hypotheses or that nonfalse consequents are entailed from true hypotheses. We shall use the terminology that hypotheses and consequents are interpreted weakly (strongly) if they are taken to be nonfalse (true). Thus, we have the four possible interpretations of $A \vdash B$, leading to four different consequence relations. These are subsequently referred to as "ss", "ws", "ww", and "sw", respectively.

It follows that ws is the most restrictive (in the sense that all sequents valid in ws are valid with the other consequence relations) and sw the least restrictive. Furthermore, ss and ww are dual in the sense that the validity of the sequent $A \vdash B$ in one is equivalent to $\neg B \vdash \neg A$ in the other. For instance, sequents like $\vdash 1/0 \neq 1/0$ are valid in sw and ww, and sequents like $1/0 = 1/0 \vdash 1/0 \neq 1/0$ are valid in sw, ww and ss (assuming division by zero is undefined and that $=$ is strict); however, none of them are valid in ws. And $\vdash A \Rightarrow A$ is not valid in ws and ss with Kleene's implication operator. With his interpretation, $A \Rightarrow B$ is true when A is false or B is true, is false when A is false and B is true, and is undefined otherwise. Other implication operators may be defined, but are not natural for computer science reasoning, and will not be considered.

As examples of logics of the different kinds we have: ss [2,4], called LPE; sw [11,12,9], called PFOL in the latter; ww [13,15], called WL; and ws [7,16]. The four consequence relations defined above are the ones most commonly identified by authors [9,14]. Other consequence relations can be defined, and more possibilities are discussed in [1,16]. In [1], one searches for a mathematically natural partial logic, restricting oneself to reflexive and transitive consequence relations, whereas the implication operator need not be as suggested by Kleene. In contrast, we restrict the implication operator, as explained, but not the consequence relation. In fact, the consequence relation of sw is not transitive, and that of ws is not reflexive.

For consequence relations that give the same interpretation of hypotheses and consequents, it is possible to define natural deduction [17] predicate calculi (in the style of e.g. [2, 15], where hypotheses occur only in premises), but for consequence relations that do not have this property, it may be necessary to use proof rules with premises as well as conclusions expressed by means of sequents. For instance, consider the following two versions of "modus ponens":

$$\frac{\vdash A \Rightarrow B \qquad \vdash A}{\vdash B}$$

and

$$\frac{\vdash A \Rightarrow B}{A \vdash B}.$$

Only the latter rule is sound in sw. Both are sound in ws and is ss, whereas neither is sound in ww. In particular, in sw one may not conclude $\vdash B$ from $\vdash A$ and $A \vdash B$.

The consequence relation of ws preserves the classical duality between the logical implication and the consequence relation, the *duality principle*, i.e. the validity of $A \vdash B$ is equivalent to that of $\vdash A \Rightarrow B$. In contrast, ww and ss do not satisfy this equivalence. It turns out that all classical rules of many sorted logic are sound in ws without any modification! Thus, ws provides reasoning closer to classical logic than the other consequence relations.

Also from an intuitive point of view, ws appears attractive: In ws nothing can be proved from or about undefinedness since an undefined hypothesis gives no information and may be ignored, whereas an undefined consequent is impossible to prove and may be replaced by false. An undefined formula occurring in (the hypothesis or in the consequent of) a sequent may be replaced by anything without losing validity.

We will investigate below the advantages of ws further, and we will discuss a formal system for ws logic (WSL).

Correspondence to the LK-calculus

Consider sequents where the hypothesis consists of a list of formulas (interpreting commas as $\wedge$'s, with true as default), and where the consequent consists of a list of formulas (interpreting commas as $\vee$'s, with false as default). The duality principle may not be formalized as: $\Gamma_1, A \vdash B, \Gamma_2$ is equivalent to $\Gamma_1 \vdash A \Rightarrow B, \Gamma_2$, which again is equivalent to $\Gamma_1 \vdash \neg A, B, \Gamma_2$ (letting indexed Γ's denote arbitrary lists of formulas, possibly empty).

As stated, WSL has the property that strongly true consequents are entailed from the hypothesis. Thus, $\vdash A, \neg A$ is not valid, reflecting the gap in truth values, but $\vdash \neg\Delta[A], A, \neg A$ is valid, where $\Delta[A]$ is the condition for the definedness of A. By the duality principle, this may be rephrased as: $A \vdash A$ is not valid, but $\Delta[A], A \vdash A$ is valid. The latter (in one of its formulations) is the logical axiom (schema) of WSL, which replaces the trivial sequent $A \vdash A$ of classical logic.

As a consequence of the duality principle, we have that the sequent $\Gamma_1, A \vdash \Gamma_2$ is equivalent to $\Gamma_1 \vdash \neg A, \Gamma_2$, and $\Gamma_1 \vdash A, \Gamma_2$ is equivalent to $\Gamma_1, \neg A \vdash \Gamma_2$. Thus, formulas in a sequent may be moved around as in classical logic. As a direct consequence, a sequent may be rewritten as one without hypotheses. A sound and complete set of proof rules for such sequents is quite simple, and a sound and complete set of proof rules for sequents with hypotheses is easily derived. It turns out that the classical rules of LK-calculus [10] are sound in ws without any modification. The requirement that substitutions must be well-defined may be expressed through the typing premises needed in the many-sorted version of LK-calculus. Together with the WSL axiom and rules for well-definedness, they form a sound and complete system. WSL without logical axioms and without the well-definedness operator is equivalent to many-sorted LK-calculus without logical axioms. The LK-calculus without logical axioms is also considered by Stärk [18] (who pointed out this fact to us).

The symbol Δ will be defined by structural induction over the language of WSL. This is possible because we will allow (nonlogical) function symbols to range over only strict functions. (With a more complicated definition of Δ, it is possible to let functions range over monotonic functions [7].) To achieve this, we have a restrictive definition of "standard structures" which results in a limited expressive power compared to other logics [4, 13, 9]. We claim that the expressive power is sufficient in computer science applications, where one is interested in limiting the use of nonmonotonic functions (since they, in general, are nonexecutable) to a few essential operators, such as Δ, strong equality and the approximation relation, which are useful for reasoning about monotonic functions (corresponding to implemented programs). In WSL, these nonmonotonic operators can be constructively defined by means of the Δ-operator.

The resulting WSL system is simple, and many interesting properties of WSL can be established by relatively simple modifications of the proofs for similar properties of classical logic. As such, we find WSL a promising candidate for a three-valued logic, since the main achievement of logic must be to define systems that are as simple as possible and yet have a sufficient expressive power.

Relation to other three-valued logics

The calculus we suggest has been discarded by others without further investigation because of the invalidity of the trivial sequent, i.e. $A \vdash A$. Against this criticism, we can argue that the trivial sequent is not so obvious in a three-valued setting. For instance, is it desirable that $0/0=1 \vdash 0/0=0$ is valid? Since this is equivalent to the trivial sequent $\perp \vdash \perp$ (assuming division by zero is undefined and $=$ strict), such sequents are valid in ww, ss and sw!

Furthermore, the "deduction theorem" and "modus ponens"[1] are sound in WSL, but not in LPF, WL, and PFOL. These rules may be formalized as

$$(\Rightarrow\text{-I}) \quad \frac{\Gamma_1, A \vdash \Gamma_2, B}{\Gamma_1 \vdash \Gamma_2, A \Rightarrow B}$$

$$(\Rightarrow\text{-E}) \quad \frac{\Gamma_1 \vdash \Gamma_2, A \Rightarrow B \qquad \Gamma_3 \vdash \Gamma_4, A}{\Gamma_1, \Gamma_3 \vdash \Gamma_2, \Gamma_4, B}$$

In LPF the rule $\Rightarrow$-I has the side-condition that A must be defined, and in WL and PFOL the rule $\Rightarrow$-E has the side-condition that A must be defined. A further problem with PFOL is that the cut rule does not hold without a similar side-condition. The presence of such side-conditions violates the symmetry of the calculus.[2] WL and PFOL have basically[3] the same expressive power as WSL, but the expressive power of LPF covers also nonmonotonic operators (since Δ is primitive in LPF). Nonmonotonicity necessitates a side-condition on the rule:

$$(\exists\text{-I}) \quad \frac{\Gamma_1 \vdash \Gamma_2, A[t/x]}{\Gamma_1 \vdash \Gamma_2,\ \exists x\colon T \cdot A}$$

(where $[t/x]$ denotes substitution of x by t), namely $\Gamma_1 \vdash \Gamma_2, \Delta_T t$. It is not sufficient that t is a well-formed term of type T. For instance, from $\vdash \neg\Delta[\perp]$ one may not conclude that $\vdash \exists x\colon T \cdot \neg\Delta[x]$.

[1] Here we mean modus ponens in the form of $\Rightarrow$-E and not in the form $A, A \Rightarrow B \vdash B$ considered in [9]. As pointed out there, the latter version is not valid in WSL.

[2] To be fair, the problem is not as intricate for the basic calculus of WL as it is for LPF and PFOL. The $\Rightarrow$-E rule is not part of the basic calculus needed to establish completeness. A pure calculus for WL would be exactly as the one for classical logic, but the problem is that one would (or could after some small modifications of the interpretation of the $\exists$-quantifier) have no information at all about undefinedness, because of the weak interpretation! This problem appears when one wants to use nonlogical axioms, cf. the $\Rightarrow$-E rule.

[3] In WL and PFOL, function symbols are interpreted as strict functions, but in both logics the nonmonotonic definedness operator, represented by Δ in this paper, is considered primitive.

By the requirement that free variables range over defined values only, and by the inductive definition of definedness, we avoid the introduction of nonmonotonic connectives in the base logic. However, essential nonmonotonic connectives may be defined constructively by means of the well-definedness operator. Thus, nonmonotonic formulas may be transformed into monotonic ones. One may extend WSL by taking the nonmonotonic connectives as logical symbols. The constructive relationship to the monotonic part of the logic may be exploited to derive sound and complete rules for the extended logic.

A more comprehensive comparison of WSL with other logics may be found in [16].

3. Classification of terms and formulas

In this section we first formalize the language of the logic and its interpretation. We define the well-definedness operator by structural induction. We end the section by some more formal considerations about the expressive power of the logic.

Syntax

WSL is many-sorted and the language is defined for a finite number $S, T, \ldots$ of different sorts. However, the sorts do not play an essential role in this paper. Formulas are always taken to be of sort *Bool* (boolean). The language is defined by logical and nonlogical symbols. The (minimal set of) logical symbols consists of true (true), undefined ($\perp$), negation ($\neg$), conjunction ($\wedge$), and, for each sort T, a universal quantifier ($\forall_T$) and an equality relation ($=_T$). The nonlogical symbols consist of a countable number of

- function symbols ($f : T \rightarrow S$),
- a characteristic predicate $\boldsymbol{D}_f : T \rightarrow Bool$ for each f,
- variables (x_T) for each sort T.

(Functions with co-domain *Bool*, i.e. predicates, will often be denoted with a P instead of an f.) The characteristic predicate of f will be used to express when f is well-defined, such that $\boldsymbol{D}_f(t)$ is true when $f(t)$ is well-defined for well-defined t. The characteristic predicate of a total function is true. All characteristic predicates are by definition total. (Therefore, the characteristic predicate of a characteristic predicate is not needed.) It is also strict and, thus, monotonic – in contrast to Δ.

The classes of well-formed terms and well-formed formulas are defined as usual. Terms and formulas involving typing conflicts are not considered well-formed. A formula or term is closed if it does not contain any free variables (in the usual sense). Substitution is defined as usual: $A[t/x]$ means that all free occurrences of x are replaced by t (renaming bound variables in A in order to avoid name clashes with free variables of t).

Monotonic extensions of the language

Other logical symbols may be introduced by the following abbreviations:

$\mathsf{false} \equiv \neg\,\mathsf{true}$	
$\exists x\colon T\cdot A \equiv \neg \forall x\colon T\cdot \neg A$	(existential quantification)
$A \vee B \equiv \neg(\neg A \wedge \neg B)$	(disjunction)
$A \Rightarrow B \equiv (\neg A) \vee B$	(implication)
$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$	(bi-implication).

It turns out that $A \Leftrightarrow B$ is equivalent to $A = B$.

A nonstrict boolean conditional can be introduced by the following abbreviation:

$$(\mathsf{if}\ A\ \mathsf{then}\ B\ \mathsf{else}\ C) \equiv (A \wedge (A \Rightarrow B)) \vee (\neg A \wedge (\neg A \Rightarrow C)).$$

Note that $(A \wedge \neg B) \vee (\neg A \wedge B)$ is total and strict with respect to A and B, and gives rise to an exclusive-or operator.

Standard structures

A standard structure for the language L with sorts $S, T, \ldots$ is defined as follows. Each sort $S, T, \ldots$ is assigned a nonempty, countable set of objects $\mathsf{S}, \mathsf{T}, \ldots$ Furthermore, $\mathsf{S}^{\perp}$ (S-lifted) is equal to $\mathsf{S} \cup \{\perp_S\}$, where $\perp_S$ is a distinguished "undefined" object for each S. In particular, $\mathsf{Bool}^{\perp} = \{\mathsf{true}, \mathsf{false}, \perp_{Bool}\}$.

A structure over L is defined as the tuple

$$\langle \{\mathsf{S}^{\perp}, \mathsf{T}^{\perp}, \ldots\}, F, [\![\ ]\!] \rangle,$$

where

F maps each function symbol f to strict function $\mathsf{f}\colon \mathsf{T}^{\perp} \to \mathsf{S}^{\perp}$ and maps each characteristic predicate symbol $\boldsymbol{D}_f$ to a strict and total function $\boldsymbol{D}_{\mathsf{f}}\colon \mathsf{T}^{\perp} \to \mathsf{Bool}^{\perp}$, such that $\boldsymbol{D}_{\mathsf{f}}$ is true whenever f is defined and false otherwise.

$[\![\ ]\!]_\sigma$ classifies any term of type T (including *Bool*) into a value of $\mathsf{T}^{\perp}$ under the "assignment" (substitution) σ of the free variables in the term (defined below).
For each variable x of type T, an assignment σ assigns a value in T, denoted as $x\sigma$. The classification of closed terms does not depend on σ.

Classification of terms and formulas

The classification of formulas is based on Kleene's extended interpretations

$[\![x]\!]_\sigma = x\sigma$	for variable x
$[\![f(t)]\!]_\sigma = (F(f))([\![t]\!]_\sigma)$	for nonlogical functions (and characteristic predicates) f

$$[\![t_1 =_T t_2]\!]_\sigma = \begin{cases} \mathsf{true} & \text{if } [\![t_1]\!]_\sigma, [\![t_2]\!]_\sigma \in \mathsf{T} \text{ and } [\![t_1]\!]_\sigma = [\![t_2]\!]_\sigma, \\ \mathsf{false} & \text{if } [\![t_1]\!]_\sigma, [\![t_2]\!]_\sigma \in \mathsf{T} \text{ and } [\![t_1]\!]_\sigma \neq [\![t_2]\!]_\sigma, \\ \bot_{Bool} & \text{otherwise.} \end{cases}$$

$$[\![\neg A]\!]_\sigma = \begin{cases} \mathsf{true} & \text{if } [\![A]\!]_\sigma = \mathsf{false}, \\ \mathsf{false} & \text{if } [\![A]\!]_\sigma = \mathsf{true}, \\ \bot_{Bool} & \text{otherwise.} \end{cases}$$

$$[\![A \wedge B]\!]_\sigma = \begin{cases} \mathsf{true} & \text{if } [\![A]\!]_\sigma = \mathsf{true} \text{ and } [\![B]\!]_\sigma = \mathsf{true}, \\ \mathsf{false} & \text{if } [\![A]\!]_\sigma = \mathsf{false} \text{ or } [\![B]\!]_\sigma = \mathsf{false}, \\ \bot_{Bool} & \text{otherwise.} \end{cases}$$

$$[\![\forall x\colon T \cdot A]\!]_\sigma = \begin{cases} \mathsf{true} & \text{if, for all } \mathsf{c} \in \mathsf{T},\ [\![A]\!]_{\sigma + [\mathsf{c}/x]} = \mathsf{true}, \\ \mathsf{false} & \text{if, for some } \mathsf{c} \in \mathsf{T},\ [\![A]\!]_{\sigma + [\mathsf{c}/x]} = \mathsf{false}, \\ \bot_{Bool} & \text{otherwise.} \end{cases}$$

$$[\![\mathsf{true}]\!]_\sigma = \mathsf{true},$$

$$[\![\bot]\!]_\sigma = \bot,$$

where $\sigma_1 + \sigma_2$ denotes overwriting σ_1 by σ_2.

Validity of sequents and soundness of inference rules

A sequent $\Gamma_1 \vdash \Gamma_2$ is valid iff for all assignments σ, there is some $A \in \Gamma_1$ such that $[\![A]\!]_\sigma = \mathsf{false}$ or some $A \in \Gamma_2$ such that $[\![A]\!]_\sigma = \mathsf{true}$. A sequent is invalid if it is not valid. An inference rule is sound if validity of all the premises implies validity of the conclusion.

Well-definedness

The Δ-symbol is a metasymbol that can be eliminated by repeated application of the following rewrite rules:

$$\Delta[A \wedge B] \equiv (\Delta[A] \wedge \Delta[B]) \vee (\Delta[A] \wedge \neg A) \vee (\Delta[B] \wedge \neg B)$$

$$\Delta[\neg A] \equiv \Delta[A]$$

$$\Delta[\forall x\colon T \cdot A] \equiv (\forall x\colon T \cdot \Delta[A]) \vee (\exists x\colon T \cdot (\Delta[A] \wedge \neg A))$$

$$\Delta_S[f(t)] \equiv \Delta_T[t] \wedge \boldsymbol{D}_f(t) \quad (\text{where } f\colon T \rightarrow S)$$

$$\Delta[\boldsymbol{D}_f(t)] \equiv \Delta_T[t] \quad (\text{where } \boldsymbol{D}_f\colon T \rightarrow Bool)$$

$$\Delta[t_1 =_T t_2] \equiv \Delta_T[t_1] \wedge \Delta_T[t_2]$$

$$\Delta_T[c_T] \equiv \mathsf{true} \quad \text{(for all constants of type } T)$$

$$\Delta_T[x_T] \equiv \mathsf{true} \quad \text{(for all variables of type } T)$$

$$\Delta[\bot] \equiv \mathsf{false}$$

Any formula involving a Δ-symbol can always be replaced with a monotonic equivalent. In this sense there is a part of the logic that is monotonic and any formula in the logic can be rewritten to a normal form, inside this monotonic part.

The simple definition of definedness over the standard structures gives rise to some nice closure properties: by structural induction over the language, it can be justified that the above definitions are both necessary and sufficient requirements for the definedness of a formula or a term, in the sense that for any structure, $[\![A]\!]_\sigma \neq \bot_{Bool}$ iff $[\![\Delta[A]]\!]_\sigma = \mathsf{true}$. Similarly, it holds for terms, i.e. $[\![t]\!]_\sigma \in \mathsf{T}$ iff $[\![\Delta_T[t]]\!]_\sigma = \mathsf{true}$ for any term of type T. Note that these properties depend on the requirement that each function and predicate is associated with a characteristic predicate.

Furthermore, it follows that $[\![\Delta[\Delta[t]]]\!]_\sigma = \mathsf{true}$ for any term t. (This can be proved by structural induction over the language.) Thus, we can accept the following axiom as consistent with the above system:

$$\vdash \Delta[\Delta[t]].$$

Note that a total and strict and-operator may be defined as a nonlogical function (predicate) *and* with the nonlogical axiom $\vdash \forall x, y\colon Bool \cdot and(x,y) = (x \wedge y)$.

Other nonmonotonic operators

Nonmonotonic connectives like strong equality and an approximation relation can be defined constructively by means of the well-definedness operator:

$$A \equiv B \;\equiv\; \Delta[A] = \Delta[B] \wedge (\Delta[A] \Rightarrow (A = B)) \quad \text{(strong equality)},$$

$$A \sqsubseteq B \;\equiv\; \Delta[A] \Rightarrow (\Delta[B] \wedge (A = B)) \quad \text{(approximation)}.$$

Examples of strong equalities are already given above.

On expressive completeness

Cheng [4] proves his basic set of connectives expressive complete for all truth-valued n-ary functions. However, as already discussed, we are interested only in the monotonic truth-valued functions.

Theorem 3.1 (Expressive completeness). *L is expressively complete for any n-ary monotone function in* $\mathsf{Bool}_\bot^n \to \mathsf{Bool}_\bot$.

Proof (*sketch only*). Blamey [3, Section 4.1] proves that the set $\{\neg, \vee, \wedge, \divideontimes, \mathsf{true}, \mathsf{false}\}$ is expressively complete (in the above sense). Interjunction can be defined in WSL as

$$A \divideontimes B \equiv ((A \wedge B) \vee (A \wedge \bot) \vee (B \wedge \bot)). \qquad \square$$

The expressive power of WSL could be increased by taking the Δ-symbol as a primitive, and by skipping the requirements of the basic functions to be strict. This modification would leave WSL with an expressive power equal to that of LPF, and still its basic system would be simpler than those discussed in Section 2. Apart from the problem with the $\exists$-I rule for LPF, none of the asymmetries of the logics [2, 9, 13] discussed in Section 2 have anything to do with their additional expressive power, and the description of the Δ-operator given above could be adopted in those logics as well. The asymmetries are due to the choice of consequence relation.

4. Minimal proof system

In the minimal system we consider sequents with empty left-hand sides, and we omit the sequent symbol in the proof rules. It is understood that a formula of the form $\Delta[t]$ has been reduced to its normal form. The rules given below are sufficient to establish the completeness of the system with equality:

Logical axioms

$$(\mathrm{Ax}) \quad \frac{}{\neg\Delta[A], A, \neg A}$$

The Δ-symbol is a metasymbol that can be eliminated by applying the definitions of Section 3.

Logical rules

$$(\neg\neg) \quad \frac{\Gamma, A}{\Gamma, \neg\neg A}$$

$$(\mathrm{Conj}) \quad \frac{\Gamma, A \qquad \Gamma, B}{\Gamma, A \wedge B}$$

$$(\text{Neg-Conj}) \quad \frac{\Gamma, \neg A, \neg B}{\Gamma, \neg(A \wedge B)}$$

$$(\mathrm{All}) \quad \frac{\Gamma, A}{\Gamma, \forall x\colon T \cdot A} \quad x \text{ (of type } T\text{) not free in } \Gamma \text{ (eigenvariable condition)}$$

$$(\text{Neg-All}) \quad \frac{\Gamma, \neg A[t/x]}{\Gamma, \neg\forall x\colon T \cdot A} \quad t \text{ of type } T$$

Those familiar with LPF [4] will remember that usually t in "Neg-All" (also known as ∃-I, cf. Section 2) is required to be well-defined. This is not necessary here, because in the case where t is undefined, it can be replaced with any other term of that sort without changing anything. It is exactly at this point that we can see what we gain by considering only the monotonic cases. If we increased the expressive power as discussed in the end of Section 3, then we would need to add a side-condition $\Delta_T[t]$ to the "Neg-All" rule, thereby destroying the symmetry of the basic calculus.

Equality

The basic equality relation is strict and monotonic (which can be seen from the definition of standard structures):

$$(=)\quad \frac{}{\neg\Delta_T[t],\, t=t}$$

This axiom has a definedness condition similar to that of the trivial sequent. In WSL all axioms must be well-defined, or have conditions ensuring definedness:

$$(=\text{-subst})\quad \frac{\Gamma, s=t \qquad \Gamma, A[s/x]}{\Gamma, A[t/x]}$$

Other rules

$$(\text{false})\quad \frac{\Gamma, \neg\text{true}}{\Gamma}$$

$$(\text{Weak})\quad \frac{\Gamma}{\Gamma, A}$$

$$(\text{Cont})\quad \frac{\Gamma, A, A}{\Gamma, A}.$$

Theorem 4.1 (Soundness and completeness). *Let Γ be a sequent over L. Γ is provable iff Γ is valid.*

The completeness and soundness of the system is proved in the usual way [8], and we omit the details.

When nonlogical axioms are introduced, elimination rules are needed as well as the following "well-definedness" rule:

$$(\text{Def})\quad \frac{\Gamma, A}{\Gamma, \Delta A}.$$

(As mentioned earlier, the elimination rules are the classical ones, but ∀-elimination must require well-defined substitutions.)

5. Some properties of WSL

In this section we show that WSL without equality has the cut elimination property and that WSL supports rewriting in the sense of "folding" and "unfolding" definitions. Both these properties are essential, but for different reasons. The cut elimination property ensures that the logic is symmetric, and gives rise to simple decision procedures. Rewriting is essential for reasoning about nonlogical axioms used for defining programs, etc.

Cut elimination

The cut rule is not part of the minimal proof system (and if it was, it could always be eliminated):

$$\text{(Cut)}\quad \frac{\Gamma_1, A \qquad \Gamma_2, \neg A}{\Gamma_1, \Gamma_2}$$

Theorem 5.1 (Cut elimination). *Any proof in the basic system without equality using cuts can be transformed to a cut-free proof.*

Proof (*sketch only*). As remarked in [18], any proof in the basic system without equality using cuts can be reduced to a proof with only atomic cuts. This can be proved in the usual way by pushing atomic cuts up into the nodes of the proof trees, and by eliminating all other cuts in the usual way. This is possible because the basic proof rules are equivalent to the similar proof rules in the LK-calculus (refer e.g. [10]). Thus, we will always end up with cuts consisting only of axioms, and they are redundant. □

Rewriting

The =-subst rule can be changed into a more convenient form, which can be used for "folding" and "unfolding" definitions (formalizing the use of ≡ in Section 3).

$$(\equiv\text{-subst})\quad \frac{\Gamma, s \equiv t \qquad \Gamma, A[s/x]}{\Gamma, A[t/x]}.$$

The approximation relation ("less-than-or-equally-defined-as") is practically useful for formulation of nonlogical axioms. For instance, the axiom $x/x \sqsubseteq 1$ is consistent with the intented semantics, whereas $x/x \equiv 1$ (or $x/x = 1$) is not (it would follow that division is total). Since rewrite rules, typically, have right-hand sides better defined than the left-hand sides, the approximation relation gives rise to unconditional rewrite rules. With such rules, rewriting of monotonic terms may be performed as usual:

$$(\sqsubseteq\text{-subst})\quad \frac{\Gamma, s \sqsubseteq t \qquad \Gamma, A[s/x]}{\Gamma, A[t/x]}.$$

Note that ≡ satisfies $t \equiv t$, and $\sqsubseteq$ satisfies $t \sqsubseteq t$ and $\perp \sqsubseteq t$.

6. Final remarks

WSL offers reasoning close to classical logic in the sense that the classical proof rules are preserved. In particular, it is appealing that the symmetry of classical logic is not destroyed by adding definedness premises in the rules.

We believe WSL handles the gap in truth values, resulting from undefined terms, where it is most natural to handle it; namely, at the level of the logical axioms. Thus, by avoiding nonmonotonicity and accepting that $A \vdash A$ does not hold, we have got a calculus which is strong enough to ensure strong validity of consequents and at the same time is simple and elegant. The constructive definition of nonmonotonic operators enables nonmonotonic formulas to be reduced to monotonic ones.

As for the restricted interpretation of functions, the full expressive power offered by some three-valued logics does not seem to be needed in computer science applications, as long as reasoning about strong equality, approximations and well-definedness is possible. Although we have required functions to be strict, this requirement can be relaxed to just a requirement for monotonicity; cf. [7]. Since executable functions and constructs in most programming languages are monotonic, these may be modeled by monotonic functions.

In particular, reasoning about recursively defined functions is described in [16]. One may reason about approximations of recursive functions without touching well-definedness issues or characteristic predicates. However, the characteristic predicates simplify reasoning about well-definedness issues. Applications of WSL to other areas of computer science, such as abstract data types, term rewriting and Hoare logic, are demonstrated in [6].

Acknowledgment

The authors are greatly indebted to Ole-Johan Dahl for motivation and for fruitful discussions on WSL. Anne B. Salvesen has provided useful feedback through careful reading of a draft version.

References

[1] A. Avron, Natural 3-valued logics–characterization and proof theory, *J. Symbolic Logic* **56** (1991) 276–294.

[2] H. Barringer, J.H. Cheng and C.B. Jones, A logic covering undefinedness in program proofs, *Acta Inform.* **21** (1984) 251–269.

[3] S. Blamey, Partial logic, in: *Handbook of Philosophical Logic, Vol. III* (1986) 1–70.

[4] J.H. Cheng, A logic for partial functions, Tech. Report Series UMCS-86-7-1, Department of Computer Science, University of Manchester, 1986.

[5] J.H. Cheng and C.B. Jones, On the usability of logics which handle partial functions, in: C. Morgan and J.C.P. Woodcock, eds., *Proc. 3rd Refinement Workshop* (Springer, Berlin, 1991).

[6] O.-J. Dahl, *Verifiable Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1992).

[7] O.-J. Dahl and O. Owe, Formal development with ABEL, Lecture Notes in Computer Science, Vol. 552 (Springer, Berlin, 1991) 320–362.
[8] M. Elvang-Gøransson, Some properties of WSL, 1991-6, Department of Informatics, University of Oslo, 1991, preprint 1991-6.
[9] A. Gavilanes-Franco and F. Lucio-Carrasco, A first order logic for partial functions, *Theoret. Comput. Sci.* **74** (1990) 37–69.
[10] J.-Y. Girard, Y. Lafont and P. Taylor, *Proofs and Types* (Cambridge Univ. Press, Cambridge 1989).
[11] A. Hoogewijs, On a formalization of the non-definedness notion, *Z. Logik Grundlag. Math.* **25** (1979) 213–217.
[12] A. Hoogewijs, A partial predicate calculus in a two-valued logic, *Z. Logik Grundlag. Math.* **29** (1983) 239–243.
[13] M. Holden, Weak logic theory, *Theoret. Comput. Sci.* **79** (1991) 295–321.
[14] B. Konikowska, A. Tarlecki and A. Blikle, A three-valued logic for software specification and validation, Lecture Notes in Computer Science, Vol. 328 (Springer, Berlin, 1988) 218–242.
[15] O. Owe, An approach to program reasoning based on a first order logic for partial functions, Report CS-081, Department of EECS, University of California, San Diego, 1984.
[16] O. Owe, Partial logics reconsidered: a conservative approach, Research Report 155, Department of Informatics, University of Oslo, 1991; *Formal Aspects of Comput.* **5** (1993), to appear.
[17] D. Prawitz, *Natural Deduction* (Almqvist & Wiksell, Stockholm, 1965).
[18] R.L. Stärk, A complete axiomatization of the three-valued completion of logic programs, *J. Logic and Comput.* **1** (1991) 811–834.