A three-valued logic for software specification and validation

By Beata Konikowska, Andrzej Tarlecki, Andrzej Blikle
June 1988

This is a very early preprint of our paper later published under the same title in VDM'88, VDM: The Way Ahead, Proc. 2nd, VDM-Europe Symposium, Dublin 1988, Lecture Notes of Computer Science, Springer Verlag 1988, pp. 218-242

PRACE IPI PAN • ICS PAS REPORTS

Beata Konikowska,
Andrzej Tarlecki, Andrzej Blikle

A three-valued logic for software specification and validation

635

June 1988

WARSZAWA

INSTYTUT PODSTAW INFORMATYKI POLSKIEJ AKADEMII NAUK
INSTITUTE OF COMPUTER SCIENCE POLISH ACADEMY OF SCIENCES
00-901 WARSAW, P.O. Box 22, POLAND

Beata Konikowska, Andrzej Tarlecki, Andrzej Blikle

A three-valued logic for software specification and validation

Tertium tamen datur

635

Warsaw, June 1988

Pracę zgłosił Antoni Mazurkiewicz

Mailing address: Project MetaSoft
Institute of Computer Science
Polish Academy of Sciences
PKiN, P.O.Box 22, 00-901 Warsaw

ISSN 0138-0648

Printed as a manuscript
Na prawach rękopisu

Nakład 350 egz. Ark. wyd. 3,00; ark. druk. 2,25.
Papier offset. kl. III, 70 g, 70 x 100. Oddano do
druku w lipcu 1988 r. WDN Zam. nr 371/88.

Abstract • Streszczenie • Содержание

Different calculi of partial or three-valued predicates have been used and studied by several authors in the context of software specification, development and validation. This paper offers a critical survey on the development of three-valued logics based on such calculi.

In the first part of the paper we review two three-valued predicate calculi, based on, respectively, McCarthy's and Kleene's propositional connectives and quantifiers, and point out that in a three-valued logic one should distinguish between two notions of validity: *strong validity* (always true) and *weak validity* (never false). We define in model-theoretic terms a number of consequence relations for three-valued logics. Each of them is determined by the choice of the underlying predicate calculus and of the weak or strong validity of axioms and of theorems. We discuss mutual relationships between consequence relations defined in such a way and study some of their basic properties.

The second part of the paper is devoted to the development of a formal deductive system of inference rules for a three-valued logic. We use the method of semantic tableaux (slightly modified to deal with three-valued formulas) to develop a Gentzen-style system of inference rules for deriving valid sequents, which we use then to obtain a sound and complete system of natural deduction rules. We have chosen to study the consequence relation determined by the predicate calculus with McCarthy's propositional connectives and Kleene's quantifiers and by the strong interpretation of both axioms and theorems. Although we find this choice appropriate for applications in the area of software specification, verification and development, we regard this logic merely as an example and use it to present some general techniques of developing a sequent calculus and a natural deduction system for a three-valued logic.

Logika trójwartościowa dla specyfikacji i weryfikacji oprogramowania

Wielu autorów studiowało rozmaite rachunki częściowych lub trójwartościowych predykatów. Niniejsza praca zawiera przegląd metod konstruowania logik trójwartościowych dla takich rachunków.

W pierwszej części pracy przypominamy dwa trójwartościowe rachunki predykatów, oparte o spójniki logiczne i kwantyfikatory McCarthy'ego i Kleene'ego, odpowiednio. Wskazujemy przy tym, że w logice trójwartościowej należy odróżniać dwa pojęcia spełnialności: mocna spełnialność (formuła zawsze prawdziwa) i słaba spełnialność (formuła nigdy fałszywa). Dla logik trójwartościowych definiujemy w terminach teorii modeli szereg relacji konsekwencji, określanych jednoznacznie przez wybór rachunku predykatów oraz mocnej albo słabej spełnialności aksjomatów i twierdzeń. Omawiamy wzajemne związki między relacjami konsekwencji zdefiniowanymi w ten sposób oraz podstawowe własności tych relacji.

Druga część pracy jest poświęcona opracowaniu formalnego systemu dedukcyjnego pewnej logiki trójwartościowej. Stosując metodę tabelek semantycznych (nieco zmodyfikowaną dla potrzeb logiki trójwartościowej) budujemy Gentzenowski system wyprowadzania poprawnych sekwentów, który wykorzystujemy z kolei dla zdefiniowania poprawnego i pełnego systemu reguł naturalnej dedukcji. Systemy te opracowujemy dla relacji konsekwencji określonej przez rachunek predykatów ze spójnikami logicznymi McCarthy'ego i kwantyfikatorami Kleene'go oraz przez mocną spełnialność aksjomatów i twierdzeń. Chociaż uważamy, że jest to wybór odpowiedni dla potrzeb specyfikacji, weryfikacji i konstruowania oprogramowania, w tej pracy traktujemy tę logikę przede wszystkim jako przykład, który wykorzystujemy dla przedstawienia pewnych ogólnych technik konstruowania rachunków sekwentowych i systemów naturalnej dedukcji dla logik trójwartościowych.

Трехзначная логика для спецификации и проверки /верификации/ программирования

Многие авторы изучали различные исчисления частичных или трехзначных предикатов. Настоящая работа заключает в себе обзор методов построения /конструирования/ трехзначных логик для таких исчислений.

В первой части работы напоминаем два трехзначные исчисления предикатов основаны, как следует, на логических связях и кванторах Маккартьего и Клина.

Указуем при этом, что в трехзначной логике надо различать два понятия выполняемости: сильная выполняемость (формула всегда истинная) и слабая выполняемость (формула никогда ложная). Для трехзначных логик определяем в терминах теории моделей ряд отношений следования, однозначно диагностированных путем выбора исчисления предикатов, а также сильной или слабой выполняемости аксиом и теорем. Обсуждаем взаимные связи между отношениями следования таким способом определенных и основные свойства этих отношений.

Вторая часть работы посвящена формальной дедуктивной системе некоторой трехзначной логики. Применяя метод семантических таблиц (чуть модифицированный для потребностей трехзначной логики) создаем генценовскую систему вывода правильных следующих элементов /последовательности/, которую используем, в свою очередь, для определения корректной и полной системы естественной /натуральной/ дедукции. Эти системы разрабатываем для отношения следования определенного исчислением предикатов с логическими связями Маккартьего и кванторами Клина, а также через сильную выполняемость аксиом и теорем. Хотя по нашему мнению выбор этот является подходящим для потребностей спецификации, проверки /верификации/ и постройки программирования в данной работе подходим к этой логике прежде всего как к примеру, который используем для представления некоторых общих техник построения исчислений следующих элементов /последовательности/ и систем естественной /натуральной/ дедукции для трехзначных логик.

1. Introduction

Tertium non datur — a law of classical logic — states that every hypothesis is either true or false. Nevertheless, sentences which people use in formulating thoughts may be meaningless and therefore neither true nor false. "This paper is tall" or "$0^{-1} > 1$" are two such examples.

In every deductive reasoning we can recognize and reject meaningless sentences and therefore every hypothesis which can be proved at all can be proved using only meaningful sentences. This makes a two-valued logic an adequate tool for proving facts. The situation changes if sentences with free variables are evaluated dynamically in the process of running an algorithm. Formulas like "x is tall" or "$x^{-1} > 1$" may be true, false or undefined depending on the value of x. Of course, if an algorithm runs across an undefined formula, then it either aborts, or loops or produces an incorrect result. In order to avoid such situations we must be able to describe them formally, and therefore we have to assume that our formulas (Boolean expressions) represent either partial functions with values in the two-element set $\{tt, ff\}$ or total functions with values in a three-element set $\{tt, ff, ee\}$, where ee represents an "abstract error" or "undefinedness". It turns out that the latter solution is more convenient since it allows us to treat undefinedness in a lazy, non-strict way (cf. [Blikle 87]).

When we admit a third logical value, we have a variety of choices in formalizing and in using three-valued Boolean expressions in software specification and validation. First, there are several ways of extending the classical two-valued predicate calculus to the three-valued case. Three major candidates for such extensions have been proposed in the literature: a calculus described by S.C.Kleene [Kleene 38, 52], a calculus described by J.McCarthy [McCarthy 61], and a combination of these calculi described in [Blikle 87]. Second, for each such extension we have two major strategies of using three-valued predicates in the specification and in the validation of software. One strategy consists in constructing a special algebra of three-valued predicates which is later used at the level of a second-order two-valued logic. This strategy has been chosen in [Blikle 87] where it was applied to the McCarthy-Kleene calculus in the context of a particular software-specification language. It was discussed later in [Blikle 88] in a more general setting.

Another strategy consists in developing a three-valued formal logic. Here we have again several possibilities. First we choose the underlying predicate calculus. Besides the choice between Kleene's and McCarthy's (or yet different) propositional connectives and quantifiers, we also have to decide whether we accept non-monotone propositional connectives. Second, we choose the concept of the validity of a formula. In a three-valued logic we distinguish between *strongly valid* (always true) and *weakly valid* (never false) formulas. When we construct a logic, i.e. an axiomatic theory of a consequence relation, we can talk about strong and weak axioms (in the above sense) and strong and weak theorems. This yields four possibilities. Finally, we can develop a sequent deduction system or a natural deduction system.

The issue of three-valued logic in the context of software specification and validation has recently attracted much attention. Of the most important contributions we should mention [Hoogewijs 79, 83] where the author discussed a three-valued logic over Kleene's connectives (including a non-monotone connective) and quantifiers, with strong axioms and weak theorems. Another logic over Kleene's calculus, with a non-monotone connective, strong axioms and strong theorems has been presented in [Barringer, Cheng & Jones 84] and its applications were discussed in [Jones 86, 87]. [Hoogewijs 87] gives a comparison of the two systems. [Owe 85] develops a formal logic with weak axioms and weak theorems based on a rather unusual predicate calculus which is a mixture of strict logical

connectives, if-expressions (under the assumptions in [Owe 85] this is sufficient to define both Kleene's and McCarthy's connectives) Kleene's existential quantifier and a "weak" universal quantifier. A more complete discussion of different three-valued logics may be found in [Cheng 86].

In this paper we attempt to tackle two issues. First, we give a general comparative analysis of semantic consequence relations corresponding to various three-valued logics. In this way we come up with a number of conclusions about three-valued logics without even starting to think about the proof-theoretic machinery of deduction rules. We compare these consequence relations with each other and with the classical consequence. For instance, we point out that a three-valued consequence relation based on any of the two calculi we consider (both without non-monotone operators) with strong axioms and weak theorems, is equivalent to the classical consequence relation (Sec. 4), and we discuss where we can and where we cannot expect the rule of detachment and the deduction theorem to hold (Sec. 5).

The second objective of this paper is to develop a system of inference rules for a three-valued logic. We develop two sets of proof rules, two formal systems (a sequent calculus and a natural deduction system) for the Kleene-McCarthy logic without non-monotone operators, with strong axioms and strong theorems. We have chosen these parameters for our logic following motivation given in [Blikle 87, 88], but to a large extent we regard this logic merely as an example which we use to discuss some general techniques of developing sequent proof rules and natural-deduction proof rules for an arbitrary three-valued logic.

We have tried to make our paper possibly self-contained for a reader who is not a logician. However, we could not make it a complete tutorial. Therefore we skip some standard technical details (trying to outline all the ideas, though) which may be found in the literature.

Acknowledgements

We are grateful to A.Hoogewijs, J.H.Cheng, O.Owe, N.Haichu and D.Sannella for their comments on an early version of the paper, which helped to improve the presentation and discussion of many points here.

2. Introductory concepts

In this section we define and discuss the predicate calculi of Kleene and of McCarthy. We also define a combination of these caluli which is to be used later as a basis for our logic. Let us start by introducing some notation.

If A and B are sets, then by $A \to B$ we denote the set of all total functions from A to B. By $f\colon A \to B$ we denote the fact that f is a total function from A to B. $g\colon A \to B \to C$ abbreviates $g\colon A \to (B \to C)$. Instead of $f(a)$ we write $f.a$ and instead of $(g.a).b$ we write $g.a.b$. For any $a \in A$, $b \in B$ and $f\colon A \to B$, by $f[b/a]$ we denote a modification of f defined as follows:

$$f[a/b].x = \text{ if } x = a \text{ then } b \text{ else } f.x$$

Let

$$Boolerr = \{tt, ff, ee\}$$

be our three-element set of logical values ($Boolerr$ stands for $Boolean + Error$). We introduce

McCarthy's conditional operator:

$$\xrightarrow{\text{MC}}: Boolerr \times Boolerr \times Boolerr \to Boolerr$$

where for any $a, b, c \in Boolerr$,

$$a \xrightarrow{\text{MC}} b, c \;=\; \begin{array}{ll} b & \text{for } a = tt, \\ c & \text{for } a = ff, \\ a & \text{for } a = ee. \end{array}$$

McCarthy's propositional connectives are defined as follows:

not a	$= a \xrightarrow{\text{MC}} ff, tt$	— *negation*,
a **or** b	$= a \xrightarrow{\text{MC}} tt, b$	— *disjunction*,
a **and** b	$= a \xrightarrow{\text{MC}} b, ff$	— *conjunction*,
a **implies** b	$= a \xrightarrow{\text{MC}} b, tt$	— *implication*.

In all the above definitions "=" denotes the identity in *Boolerr*.

McCarthy's connectives have four important properties:

1. They extend classical connectives, i.e. coincide with them on the logical values tt and ff. They also satisfy the classical mutual relationships such as for example:

$$a \textbf{ implies } b = (\textbf{not } a) \textbf{ or } b \tag{1}$$

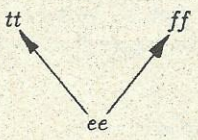
$$a \textbf{ and } b = \textbf{not}((\textbf{not } a) \textbf{ or } (\textbf{not } b)) \tag{2}$$

   However, not all classical identities hold. For example, neither of the following is true in general:

   - a **or** $b = b$ **or** a
   - a **or** (**not** a) $= tt$

2. They allow for a lazy evaluation of expressions and in that case they again coincide with classical operators. For example, if we evaluate exp_1 **or** exp_2 and if exp_1 evaluates to tt, then we can abandon the evaluation of exp_2, since independently of its value (even if it is ee) the value of the whole expression is tt.

3. They are monotone in the usual cpo over *Boolerr*:



   That is, if in a propositional expression we replace one of its current arguments by ee, then the value of that expression either remains unchanged, or becomes ee. This is in contrast with some other three-valued calculi. For example, [Hoogevijs 79, 83, 87] and [Barringer, Cheng & Jones 84] study a calculus similar to Kleene's calculus as presented below, but with a total non-monotone unary connective Δ which determines whether or not its argument is "defined" (i.e. $\neq ee$).

4. They are strict with respect to their left argument, i.e. assume value *ee* whenever their left argument is *ee*.

Properties 2 and 3 guarantee that the third logical value *ee* may be interpreted not only as an abstract error (i.e. as an error signal generated by the system), but also as true undefinedness resulting from a nonterminating computation. Due to property 4, McCarthy's Boolean expressions may be implemented on a sequential machine.

As the reader has probably already noticed, McCarthy's disjunction and conjunction are not commutative. This is the price which we have to pay for a propositional calculus which has properties 1, 2 and 3 and which is sequentially implementable at the same time. If we do not care about the latter property, we can select the propositional calculus of Kleene, where negation (**not**) is the same as above, disjunction ($\mathbf{or}_K$) is the following extension of the classical connective:

$$tt\ \mathbf{or}_K\ ee = ee\ \mathbf{or}_K\ tt = tt$$
$$a\ \mathbf{or}_K\ ee = ee\ \mathbf{or}_K\ a = ee \qquad \text{for } a \neq tt,$$

and the remaining operators are defined by identities (1) and (2). Now we have the commutativity of conjunction and of disjunction, but in order to implement expressions in the case where *ee* may correspond to a nonterminating computation, we need unbounded parallelism. Indeed, when we compute

$$exp_1\ \mathbf{or}_K\ \ldots\ \mathbf{or}_K\ exp_n$$

we have to compute all exp_i $(i = 1, \ldots, n)$ in parallel in order to determine whether one of them evaluates to *tt*.

Now, let us talk about predicates and quantifiers. In this section we discuss them on semantic, model-theoretic grounds. The corresponding formalized language of logic is defined in the next section.

Let *Ide* be an arbitrary nonempty set of *identifiers* (variables) and let *Value* be an arbitrary nonempty set of *values*. A *state* is a total function from identifiers to values; a *predicate* is a total function from states to logical values:

$$State = Ide \to Value$$
$$Predicate = State \to Boolerr$$

Logicians would have probably preferred to say *valuations* rather than *states*. We use the latter term since it is more common in the field where our logic is to be applied.

The elements of *State* will be denoted by *sta*, the elements of *Ide* by $x, y, z, \ldots$, the elements of *Value* by v, val, and the elements of *Predicate* by $p, q, \ldots$, all possibly with indices, primes etc.

In this framework quantifiers may be regarded as functions that assign predicate transformers to identifiers:

Forall: $Ide \to Predicate \to Predicate$
Exists: $Ide \to Predicate \to Predicate$

In this paper we define quantifiers after [Kleene 52] by extending the corresponding classical definitions to the three-valued case. For all $p \in Predicate$, $x \in Ide$ and $sta \in State$ we set:

$\mathbf{Forall}.x.p.sta =$ tt if for all values $val \in Value$, $p.(sta[val/x]) = tt$;
ff if for some value $val \in Value$, $p.(sta[val/x]) = ff$;
ee otherwise, i.e.
if for no value $val \in Value$, $p.(sta[val/x]) = ff$,
but for some value $val \in Value$, $p.(sta[val/x]) = ee$.

$\mathbf{Exists}.x.p.sta =$ tt if for some value $val \in Value$, $p.(sta[val/x]) = tt$;
ff if for all values $val \in Value$, $p.(sta[val/x]) = ff$;
ee otherwise, i.e.
if for no value $val \in Value$, $p.(sta[val/x]) = tt$,
but for some value $val \in Value$, $p.(sta[val/x]) = ee$.

These quantifiers satisfy de Morgan's laws and generalize Kleene's conjunction and disjunction. However, they are not the generalizations of McCarthy's respective connectives. Quantifiers which attempt to generalize McCarthy's connectives have been defined in [McCarthy 61]. In that case:

$\mathbf{Exists}_{Mc}.x.p.sta = tt$ if for some value $val \in Value$, $p.(sta[val/x]) = tt$,
and for all values $val \in Value$, $p.(sta[val/x]) \neq ee$,

all other cases are changed accordingly, and the universal quantifier is then defined by the de Morgan laws.

The Kleene definition of quantifiers is, in our opinion, more natural. Moreover, McCarthy's quantifiers are non-monotone and for a recursive p, $\mathbf{Exists}_{Mc}.x.p$ is, in general, not recursively enumerable.

In the sequel our predicate calculus, based on McCarthy's propositional connectives and Kleene's quantifiers, will be referred to as the *MK-calculus* (for "McCarthy-Kleene") and the calculus based on Kleene's propositional connectives and quantifiers as the *K-calculus*.

3. Consequence relations based on three-valued predicate calculi

A formalized logic may be regarded as an axiomatic theory of a consequence relation between formulas (we refer to [Avron 87] for an expository presentation of this point of view). In this section we define a certain scheme of such a relation (based on a model-theoretic view of three-valued predicate calculi) and we instantiate it later to different logical systems. We start from the concept of a formalized language and of its semantic interpretation. Let:

Ide be a countably infinite set of identifiers (variables),
Fun_m be a set of m-ary function symbols, $m = 0, 1, \ldots$,
$Fun = \bigcup_{m=0}^{\infty} Fun_m$,
$Pred_m$ be a set of m-ary predicate symbols, $m = 1, 2, \ldots$,
$Pred = \bigcup_{m=1}^{\infty} Pred_m$,

and let

Term	denote the set of all terms constructed in the usual way over *Fun*, *Ide* and appropriate auxiliary symbols such as parentheses, commas, etc.
Form	denote the set of all formulas constructed in the usual way over *Term*, *Pred*, appropriate auxiliary symbols and the following special logical symbols: $\neg$, $\vee$, $\wedge$, $\supset$, $\forall$, $\exists$.

The two sets *Term* and *Form* constitute our formalized logical language. Throughout the rest of the paper we will consider an arbitrary (but fixed) language of such a form. For technical reasons we assume that at least one $Pred_m$, $m = 1, 2, \ldots$, is nonempty.

It should be pointed out that in this paper we deal with one-sorted logics, rather than with many-sorted logics which are more general and more appropriate for applications to software specification and validation. This, however, is only for the technical convenience of the presentation. All the notions and results we present carry over directly to the many-sorted case.

A *logical structure*, or simply *structure* (sometimes also called a *model*, but we reserve this term for a somewhat different concept) for a given logical language consists of a nonempty set *Value* of *values* and of two families of functions:

$$F_m : Fun_m \rightarrow Value^m \rightarrow Value \qquad \text{for } m = 0, 1, \ldots$$
$$P_m : Pred_m \rightarrow Value^m \rightarrow Boolerr \qquad \text{for } m = 1, 2, \ldots$$

(Recall that $Boolerr = \{tt, ff, ee\}$ is our set of three logical values.)

We say that a structure is *classical* if for $m = 1, 2, \ldots$, for all $p \in Pred_m$ and $\vec{v} \in Value^m$, $P_m.p.\vec{v} \neq ee$.

The careful reader has probably noticed that we use the term *predicate* in this paper to name both functions mapping *states* into *Boolerr* (as introduced in the previous section) as well as predicates in the more traditional sense, i.e. functions mapping *values* into *Boolerr*. We hope that this will never lead to any ambiguity.

Observe that from the formal point of view the meanings of all function and predicate symbols are total functions. However, since the set *Boolerr* contains the "undefined element" *ee*, we can interpret our predicates as partial. The same, of course, may be applied to functions. Each time we wish to do so, we may assume that the set *Value* of values contains a special element representing "undefinedness". The only additional requirement would be that all functions and predicates are monotone w.r.t. the obvious flat ordering with the "undefined element" at the bottom. In this sense our model covers partial functions as well. We are not going to discuss this explicitly here, since contrary to the case of propositional connectives over *Boolerr*, we do not consider any specific functions or predicates. In particular, in order to keep our paper within a reasonable size, the equality predicate will not be discussed at all. Let us just point out that in the case of partial functions the requirement of monotonicity mentioned above excludes the so-called strong equality (cf. e.g. [Barringer, Cheng & Jones 84]) which is not monotone.

One more general comment is perhaps appropriate here: we have introduced in our model only one "undefined value". This means that we have identified two different kinds of undefinedness:

- the undefinedness resulting from an infinite computation, and

- the undefinedness resulting from a computable error signal (an abstract error in the sense of [Goguen 77]).

We believe that this is appropriate at the logical level, where we describe and prove properties of states, and so indirectly software properties as well (we sketch this view in more in detail Section 10). This is, of course, in contrast with the situation at the level of software semantics, where it is necessary not only to distinguish between these two kinds of undefinedness, but in fact to distinguish between different error messages and perhaps describe some operations which model the "error handling" in the software system. In our view, however, this should be modelled at the level of "data" for our logic, via the functions of the structure that describes the software system, and should not affect directly the logical part. Thus, in the set *Value* of values, but not in the set *Boolerr* of logical values, we would normally replace a single "undefined element" by a set of error elements

$$Error = \{inf, err_1, \ldots, err_n\}$$

where *inf* represents the "true undefinedness" resulting from an infinite computation and $err_1, \ldots, err_n$ represent computable error signals. Notice that from the abstract point of view the computable error signals are as much "normal elements" of the structure we describe as any other data and they should be treated as such in our theory. In particular, we would require that the functions and predicates of the structure we deal with are monotone w.r.t. the flat ordering where the "true undefinedness elelment" *inf* is at the bottom, and all the other error elements are incomparable (and incomparable with other data). This allows a natural description of "error handling" without affecting the logical issues we discuss in this paper.

For every structure we define the set of states (over this structure):

$$State = Ide \rightarrow Value$$

and the semantic function for terms:

$$T\text{:}\, Term \rightarrow State \rightarrow Value$$

(the obvious definition is omitted — $T.t.sta$ is the value of term t in the structure in state sta).

A semantic function for formulas will map them to predicates (i.e. functions from states to the set of logical values). Such a function is unambigously determined by any interpretation of the special logical symbols, i.e. by any (three-valued) predicate calculus (together with a semantic function for terms). Since in the previous section we introduced two such calculi, with every structure we associate two semantic functions for formulas:

$$F_{MK}\text{:}\, Form \rightarrow Predicate$$

determined by the MK-calculus, and

$$F_K\text{:}\, Form \rightarrow Predicate$$

determined by the K-calculus, where

$$Predicate = State \rightarrow Boolerr.$$

Thus, for example, for any formulas $\varphi_1, \varphi_2 \in Form$ and $sta \in State$:

$$F_{MK}.(\varphi_1 \vee \varphi_2).sta = F_{MK}.\varphi_1.sta \text{ or } F_{MK}.\varphi_2.sta$$
$$F_K.(\varphi_1 \vee \varphi_2).sta = F_K.\varphi_1.sta \text{ or}_K F_K.\varphi_2.sta$$

and for any formula $\varphi \in Form$ and identifier $x \in Ide$:

$$F_{MK}.(\exists x)\varphi = \text{Exists}.x.(F_{MK}.\varphi)$$
$$F_K.(\exists x)\varphi = \text{Exists}.x.(F_K.\varphi)$$

(again, a complete formal definition is omitted).

Clearly, if the considered logical structure is classical then the two semantic functions for formulas coincide, which we will make explicit by using a common name for them: in classical structures we define $F_C = F_{MK} = F_K$.

Let A be an arbitrary structure with the semantics of formulas F_{MK} and F_K, $sta \in State$ be an arbitrary state (over A) and let $\varphi \in Form$ be an arbitrary formula. The following definitions are analogous for the MK-calculus and for the K-calculus, so we give them in a schematic form for $\nabla \in \{MK, K\}$.

We say that A *strongly* ∇*-satisfies* φ *in* sta (or that φ *strongly* ∇*-holds in* $\langle A, sta\rangle$), written $\langle A, sta\rangle \models^{\nabla}_{s} \varphi$, if $F_\nabla.\varphi.sta = tt$.

We say that A *weakly* ∇*-satisfies* φ *in* sta (or that φ *weakly* ∇*-holds in* $\langle A, sta\rangle$), written $\langle A, sta\rangle \models^{\nabla}_{w} \varphi$, if $F_\nabla.\varphi.sta \neq ff$.

If A is classical then the above notions coincide (and coincide with the classical definition) and we simply say then that A *satisfies* φ *in* sta (or that φ *holds in* $\langle A, sta\rangle$).

Further, let $Ax \subseteq Form$ be an arbitrary set of formulas, which we shall refer to as axioms. We say that $\langle A, sta\rangle$ is a *strong* (respectively, *weak*) ∇*-model of* Ax, or that it *strongly* (respectively, *weakly*) ∇*-satisfies* Ax, if each $\varphi \in Ax$ strongly (respectively, weakly) ∇-holds in $\langle A, sta\rangle$. We say that $\langle A, sta\rangle$ is a *classical model of* Ax if A is classical and each $\varphi \in Ax$ holds in $\langle A, sta\rangle$.

Now we can introduce a parametric class of consequence relations:

$$\models^{\nabla}_{\beta\gamma} \subseteq 2^{Form} \times Form$$

where $\nabla \in \{C, K, MK\}$ and $\beta, \gamma \in \{s, w\}$.

We say that a formula $\varphi \in Form$ is a β-γ-∇*-consequence of* Ax, in symbols:

$$Ax \models^{\nabla}_{\beta\gamma} \varphi$$

if the formula φ γ-∇-holds in every β-∇-model of Ax[1].

(In examples we write $\varphi_1, \ldots, \varphi_n \models^{\nabla}_{\beta\gamma} \varphi$ instead of $\{\varphi_1, \ldots, \varphi_n\} \models^{\nabla}_{\beta\gamma} \varphi$.)

For each $\nabla \in \{MK, K\}$ we have four corresponding consequence relations, depending on whether we are considering strong or weak models and strong or weak theorems. In the sequel, however, we exclude w-s-∇-consequences from our considerations, since they lead to theories where an axiom does not need to be a theorem.

As to the other papers on three-valued logics, which we mentioned earlier, [Hoogewijs 79, 83] deals with s-w-K-consequence, [Barringer, Chang & Jones 84] probably with s-s-K-consequence (although

[1] We hope that this notation, although not quite formal, is understandable; for example, by "w-K-holds" we mean "weakly K-holds", by "s-MK-model" we mean "strong MK-model", etc.

in that paper the authors restrict their attention to syntactic proof rules leaving the semantics of their logic completely undefined) and [Owe 85] with *w-w-(yet different calculus)*-consequence.

β-γ-∇-consequences of an empty set of axioms are called β-γ-∇-*tautologies*. However, we do not have to mention the β parameter when dealing with tautologies, since every w-model of the empty set of formulas is also an s-model of that set and vice versa. The fact that a formula φ is a γ-∇-tautology is denoted by $\models^{\nabla}_{\gamma} \varphi$.

When we consider the classical consequence relation we may omit both the β and γ parameters, since strong satisfaction and weak satisfaction mean the same in classical structures. We write, therefore, $Ax \models^{C} \varphi$ and $\models^{C} \varphi$ with the obvious meaning.

The consequence relations introduced above may be referred to as the consequence relations of *truth*, as opposed to the consequence relations of *validity* where open formulas are always (implicitly) universally quantified (cf. [Avron 87] where this distinction is made explicit and discussed). For example, the classical consequence relation of validity is defined as follows:

$Ax \models^{C}_{validity} \varphi$ if for every classical structure A:
if for every state sta over A, $\langle A, sta\rangle$ is a model of Ax
then for every state sta' over A, formula φ holds in $\langle A, sta'\rangle$.

Clearly, the difference occurs only if we allow open formulas to be used as axioms, and so we consider it irrelevant for practical purposes. Although it seems more convenient to deal with the consequence relations of truth here, the technical analysis and results presented in the rest of the paper carry over with little modification to the case of the consequence relations of validity.

4. A comparison of β-γ-∇-consequence relations

In the previous section we have introduced a number of (apparently) different consequence relations on the same set of formulas of a formalized language. We have already mentioned some trivial facts about their mutual relationships, for example that all the consequence relations determined by the classical notion of a structure coincide. In this section we will have a closer look at the somewhat less obvious relationships between the consequence relations based on three-valued calculi.

Let us start by pointing out some facts connecting the different notions of satisfaction we consider

Fact 4.1 For any structure A and state sta over A, for any formula $\varphi \in Form$,
if $F_{MK}.\varphi.sta \neq ee$ then $F_{K}.\varphi.sta = F_{MK}.\varphi.sta$.

Proof An obvious proof by induction on the structure of φ is omitted. □

Fact 4.2 For any structure A, state sta over A and formula φ:

1. $\langle A, sta\rangle \models^{MK}_{s} \varphi$ implies $\langle A, sta\rangle \models^{K}_{s} \varphi$ (not reversible in general).
2. $\langle A, sta\rangle \models^{K}_{s} \varphi$ implies $\langle A, sta\rangle \models^{MK}_{w} \varphi$ (not reversible in general).
3. $\langle A, sta\rangle \models^{K}_{w} \varphi$ implies $\langle A, sta\rangle \models^{MK}_{w} \varphi$ (not reversible in general).

Proof All the implications follow directly from Fact 4.1. To prove that they are not reversible, consider the following formulas (throughout the paper we will use the symbol $\cong$ to denote the textual identity of formulas)

$$\varphi_1 \cong \sqrt{x} \geq 0 \vee x < 0$$
$$\varphi_2 \cong \sqrt{x} \geq 0$$

If R is the usual model of the language of arithmetic and $sta.x$ is less than 0, then φ_1 strongly K-holds, but does not strongly MK-holds in $\langle R, sta\rangle$; φ_2 weakly MK-holds, but does not strongly K-hold in $\langle R, sta\rangle$; and $\neg\varphi_1$ weakly MK-holds, but does not weakly K-hold in $\langle R, sta\rangle$. □

Lemma 4.1 Let $\nabla \in \{MK, K\}$. Consider two structures A and A', which have a common set *Value* of values and common semantics of function symbols (hence, a common semantics of terms as well). Further, assume that predicates of A' are more defined than the corresponding ones of A, i.e. that for $m = 1, 2, \ldots$, for $p \in Pred_m$ and $\vec{v} \in Value^m$, if $P_m.p.\vec{v} \neq ee$ then $P'_m.p.\vec{v} = P_m.p.\vec{v}$. Let F_∇ and F'_∇ be the semantics of formulas in A and A' respectively.

Then: for any formula $\varphi \in Form$ and state sta over A (which is a state over A' as well),

$$F_\nabla.\varphi.sta \neq ee \quad \text{implies} \quad F_\nabla.\varphi.sta = F'_\nabla.\varphi.sta$$

Proof Again, we omit a rather straightforward proof by structural induction on φ. □

It should be pointed out that the above lemma (and its consequences, like Corollary 4.1 and Theorem 4.1.4 below) holds only because all the propositional connectives and quantifiers of the calculus we use are monotone.

Corollary 4.1 Let $\nabla \in \{MK, K\}$. If a set of formulas $Ax \subseteq Form$ has a strong ∇-model then it has a classical model.

Proof Let A be a structure and let sta be a state over A. Define A^+ to be the "positive totalization" of A, i.e. a classical structure which is the same as A except that in A^+ for $m = 1, 2, \ldots$, for $p \in Pred_m$ and $\vec{v} \in Value^m$, $P^+_m.p.\vec{v} = tt$ whenever $P_m.p.\vec{v} = ee$ (and $P^+_m.p.\vec{v} = P_m.p.\vec{v}$ otherwise).

Assume now that $\langle A, sta\rangle$ is a strong ∇-model of Ax. Then by Lemma 4.1, $\langle A^+, sta\rangle$ is a classical model of Ax. □

Notice that the above corollary implies that under strong interpretation of axioms we have no way to ensure that a formula is undefined. More precisely:

Corollary 4.2 Let $\nabla \in \{MK, K\}$. For any set of formulas $Ax \subseteq Form$ and formula $\varphi \in Form$, if $Ax \models^\nabla_{ss} \varphi \wedge \neg\varphi$ then Ax has no strong ∇-model.

Proof Follows from Corollary 4.1. □

We are ready now to state the main theorem of this section, which fully describes the relationship between the consequence relations we consider.

Theorem 4.1 For any $Ax \subseteq Form$, $\varphi \in Form$, $\nabla \in \{MK, K\}$ and $\beta, \gamma \in \{s, w\}$:

1. $Ax \models^\nabla_{ww} \varphi$ implies $Ax \models^\nabla_{sw} \varphi$ (not reversible in general)
2. $Ax \models^\nabla_{ss} \varphi$ implies $Ax \models^\nabla_{sw} \varphi$ (not reversible in general)

3. $Ax \models^{\nabla}_{\beta\gamma} \varphi$ implies $Ax \models^{C} \varphi$

4. $Ax \models^{C} \varphi$ iff $Ax \models^{\nabla}_{sw} \varphi$

5. In general neither $Ax \models^{MK}_{ss} \varphi$ implies $Ax \models^{K}_{ss} \varphi$, nor $Ax \models^{K}_{ss} \varphi$ implies $Ax \models^{MK}_{ss} \varphi$. That is, the s-s-K-consequence and s-s-MK-consequence are incomparable.

6. In general neither $Ax \models^{\nabla}_{ss} \varphi$ implies $Ax \models^{\nabla}_{ww} \varphi$, nor $Ax \models^{\nabla}_{ww} \varphi$ implies $Ax \models^{\nabla}_{ss} \varphi$. That is, the s-s-∇-consequence and the w-w-∇-consequence are incomparable.

7. In general neither $Ax \models^{K}_{ww} \varphi$ implies $Ax \models^{MK}_{ww} \varphi$, nor $Ax \models^{MK}_{ww} \varphi$ implies $Ax \models^{K}_{ww} \varphi$. That is, the w-w-K-consequence and the w-w-MK-consequence are incomparable.

Proof

1. Follows directly from the fact that strong ∇-satisfaction implies weak ∇-satisfaction.

   To see the nonreversibility of the implication notice that if a formula tau is a classical tautology (e.g. $tau \cong (\forall x)(p(x) \vee \neg p(x))$) then $\neg tau$ never strongly ∇-holds, and so $\neg tau \models^{\nabla}_{sw} \varphi$ for all formulas φ, but $\neg tau$ is weakly ∇-satisfiable, and so there are formulas $\varphi \in Form$ such that $\neg tau \not\models^{\nabla}_{ww} \varphi$.

2. Follows directly from the fact that strong ∇-satisfaction implies weak ∇-satisfaction, but not vice versa in general.

3. This is a direct consequence of the fact that in classical structures both semantic functions F_{MK} and F_K coincide with the classical interpretation of formulas.

4. Because of the above, it is enough to prove that if $Ax \models^{C} \varphi$ then $Ax \models^{\nabla}_{sw} \varphi$.

   Let A be a structure and let sta be a state over it. Define A^+ to be the "positive totalization" of A as defined in the proof of Corollary 4.1.

   Assume now that $\langle A, sta\rangle$ is a strong ∇-model of Ax. Then by Lemma 4.1, $\langle A^+, sta\rangle$ is a classical model of Ax, and so by the assumption, $\langle A^+, sta\rangle$ satisfies φ. Thus, by Lemma 4.1 again, $\langle A, sta\rangle$ weakly ∇-satisfies φ.

5. Just consider the following counterexamples:

   - $(\forall x)(p(x) \vee q(x)) \models^{MK}_{ss} (\forall x)(p(x) \vee \neg p(x))$
     $(\forall x)(p(x) \vee q(x)) \not\models^{K}_{ss} (\forall x)(p(x) \vee \neg p(x))$
   - $(\forall x)p(x) \models^{K}_{ss} (\forall x)(q(x) \vee p(x))$
     $(\forall x)p(x) \not\models^{MK}_{ss} (\forall x)(q(x) \vee p(x))$

6. To see that the first implication does not hold, notice that if tau is a classical tautology then (just as in 1. above) $\neg tau \models^{\nabla}_{ss} \varphi$ for all formulas φ, although clearly there are formulas $\varphi' \in Form$ such that $\neg tau \not\models^{\nabla}_{ww} \varphi'$.

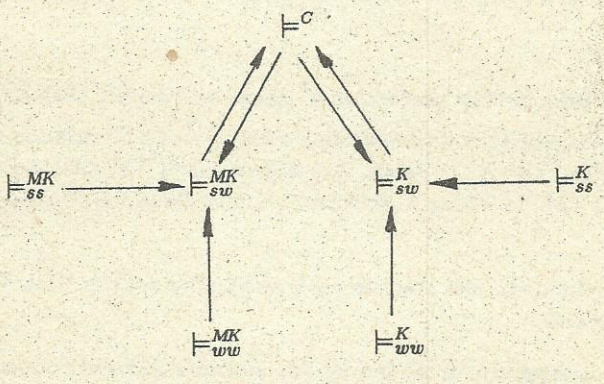
   To see that the second implication does not hold, consider weak and strong ∇-tautologies: every classical tautology is a weak ∇-tautology, but not necessarily a strong one.

7. Just consider the following counterexamples:

- $(\forall x)(p(x) \wedge q(x)) \models^{K}_{ww} (\forall x)q(x)$
  $(\forall x)(p(x) \wedge q(x)) \not\models^{MK}_{ww} (\forall x)q(x)$
- $(\forall x)(p(x) \wedge \neg p(x)) \models^{MK}_{ww} (\forall x)(p(x) \wedge q(x))$
  $(\forall x)(p(x) \wedge \neg p(x)) \not\models^{K}_{ww} (\forall x)(p(x) \wedge q(x))$

□

The above theorem may be summarized by the following picture, where the arrows represent all the inclusions (except those that follow by transitivity) between the consequence relation:



For tautologies we have the following analog of the former theorem:

Corollary 4.3 For any $f \in Form$ and $\nabla \in \{MK, K\}$:

1. $\models^{\nabla}_{s} \varphi$ implies $\models^{\nabla}_{w} \varphi$ (not reversible)
2. $\models^{\nabla}_{w} \varphi$ iff $\models^{C} \varphi$

Proof Follows from the appropriate parts of Theorem 4.1 □

Notice, however, that the first part of the above corollary holds trivially:

Fact 4.3 Let $\nabla \in \{MK, K\}$. In our logical language there are no strong ∇-tautologies. That is, for no formula $\varphi \in Form$, $\models^{\nabla}_{s} \varphi$.

Proof Consider a structure A with totally undefined predicates, i.e. such that for $m = 1, 2, \ldots$, for $p \in Pred_m$ and $\vec{v} \in Value^m$, $P_m.p.\vec{v} = ee$. By an easy induction on the structure of formula $\varphi \in Form$ we can show that for all states sta, $F_{\nabla}.\varphi.sta = ee$. □

Although s-s-∇-consequence and w-w-∇-consequence are incomparable (by Theorem 4.1.6) there is an intrinsic relationship between them:

Fact 4.4 Let $\nabla \in \{MK, K\}$. For any formulas $\varphi, \psi \in Form$

$$\varphi \models^{\nabla}_{ss} \psi \quad \text{iff} \quad \neg\psi \models^{\nabla}_{ww} \neg\varphi$$

Proof Follows from the fact that for any structure A and state sta over A

- $\langle A, sta\rangle \not\models^{\nabla}_{s} \psi$ iff $\langle A, sta\rangle \models^{\nabla}_{w} \neg\psi$
- $\langle A, sta\rangle \models^{\nabla}_{s} \varphi$ iff $\langle A, sta\rangle \not\models^{\nabla}_{w} \neg\varphi$

□

5. Some logical properties of β-γ-∇-consequence relations

By a logic over a (semantic) consequence relation $\models$ we mean a set of syntactic inference rules which define a syntactic consequence relation $\vdash$ (sometimes referred to as syntactic entailment). This new relation must be at least consistent with the former, i.e. $Ax \vdash \varphi$ must imply $Ax \models \varphi$. If the underlying formalized language is of first order (as in our case), then we should also try to make this new relation complete, i.e. such that $Ax \models \varphi$ implies $Ax \vdash \varphi$. As is known from the famous Gödel incompleteness theorem, we cannot expect a complete $\vdash$ for a second-order logic.

Although we have not yet presented a logic for any of our consequence relations, we nevertheless can investigate the properties of such logics, since if $\vdash$ is consistent and complete, then it coincides with $\models$. We consider below some basic properties of classical logic: the rule of detachment, the deduction theorem and the relationship between the consistency of a theory and the existence of a model. We check which of these properties are satisfied by our consequence relations.

It turns out that from this point of view MK-calculus and K-calculus are quite similar. Let $\nabla \in \{MK, K\}$ throughout this section. Recall that by Theorem 4.1 the s-w-∇-consequences coincide with the classical consequence. Thus, we will consider below only s-s-∇-consequence and w-w-∇-consequence.

In classical logic, one of the most frequently used inference rules is the *rule of detachment* also known as ***modus ponens***. This rule says that

$$Ax \models \varphi \supset \psi \text{ and } Ax \models \varphi \text{ implies } Ax \models \psi.$$

Theorem 5.1 The rule of detachment is satisfied by s-s-∇-consequence, but is not satisfied by w-w-∇-consequence.

Proof For s-s-∇-consequence: for any model A and a state sta, if $\varphi \supset \psi$ and φ strongly ∇-hold in $\langle A, sta\rangle$, then clearly ψ strongly ∇-holds in $\langle A, sta\rangle$.

For w-w-∇-consequence: just notice that

$$(\forall x)(p(x) \wedge \neg p(x)) \models^{\nabla}_{ww} (\forall x)(p(x)) \supset (\forall x)(q(x))$$
$$(\forall x)(p(x) \wedge \neg p(x)) \models^{\nabla}_{ww} (\forall x)(p(x))$$
$$(\forall x)(p(x) \wedge \neg p(x)) \not\models^{\nabla}_{ww} (\forall x)(q(x))$$

□

Another very important property of classical logic is the *deduction theorem* which establishes the following relationship between the logical consequence relation and implication:

$$Ax \models \varphi \supset \psi \text{ iff } Ax \cup \{\varphi\} \models \psi$$

If the deduction theorem holds, then proving that a formula ψ is a logical consequence of a finite set of axioms $\{\varphi_1, \ldots, \varphi_n\}$ may be reduced to proving that the implication $\varphi_1 \wedge \ldots \wedge \varphi_n \supset \psi$ is a tautology; and vice versa: proving that the implication $\varphi_1 \wedge \ldots \wedge \varphi_n \supset \psi$ is a tautology may be reduced to proving that ψ is a logical consequence of $\{\varphi_1, \ldots, \varphi_n\}$.

Theorem 5.2 For any set $Ax \subseteq Form$ of axioms and formulas $\varphi, \psi \in Form$:

1. $Ax \models^{\nabla}_{ss} \varphi \supset \psi$ implies $Ax \cup \{\varphi\} \models^{\nabla}_{ss} \psi$, but the implication is not reversible in general.
2. $Ax \cup \{\varphi\} \models^{\nabla}_{ww} \psi$ implies $Ax \models^{\nabla}_{ww} \varphi \supset \psi$, but the implication is not reversible in general.

Proof

1. The implication follows immediately by the rule of detachment (Th. 5.1)

   To see that it is not reversible, notice that $\varphi \models^{\nabla}_{ss} \varphi$, but $\not\models^{\nabla}_{ss} \varphi \supset \varphi$ (by Fact 4.3).

2. Assume that $Ax \cup \{\varphi\} \models^{\nabla}_{ww} \psi$, but $Ax \not\models^{\nabla}_{ww} \varphi \supset \psi$, i.e. there exists a weak ∇-model $\langle A, sta \rangle$ of Ax such that $\varphi \supset \psi$ does not weakly ∇-hold in $\langle A, sta \rangle$. By definition this means that φ strongly (and hence weakly) ∇-holds in $\langle A, sta \rangle$ and ψ does not weakly ∇-hold in $\langle A, sta \rangle$, which yields a contradiction.

   The opposite implication would entail the rule of detachment, which by Theorem 5.1 does not hold for w-w-∇-consequence.

□

For any consequence relation $\models$ and a set of axioms Ax, by a theory over Ax we mean the set

$$Cn.Ax = \{\varphi \mid Ax \models \varphi\}$$

of all the consequences of Ax. We say that Ax is *inconsistent* if there exists a formula φ such that both φ and $\neg\varphi$ belong to $Cn.Ax$. Otherwise we say that Ax is *consistent*.

In the framework of classical logic the following properties of a set Ax of formulas are equivalent:

1. Ax is consistent,
2. $Cn.Ax \neq Form$, i.e. there is a formula φ such that $Ax \not\models \varphi$,
3. Ax has a model.

Theorem 5.3 Let $Ax \subseteq Form$.

1. For s-s-∇-consequence the following properties are equivalent:
   (a) Ax is consistent,
   (b) $Cn.Ax \neq Form$,
   (c) Ax has a strong ∇-model.
2. For w-w-∇-consequence:
   (a) if Ax is consistent then $Cn.Ax \neq Form$,

(b) if $Cn.Ax \neq Form$ then Ax has a weak ∇-model,

(c) Ax may have a weak ∇-model and be inconsistent.

Proof Let $\beta \in \{w, s\}$. In both cases, if Ax is consistent then for any $\varphi \in Cn.Ax$, $\neg\varphi \notin Cn.Ax$ and so $Cn.Ax \neq Form$. Next, if Ax has no β-∇-models then by our definition $Cn.Ax = Form$. In fact for weak ∇-models this implication holds trivially, since any set of axioms has a weak ∇-model (take the structure with totally undefined predicates). To complete the proof of the implications that hold notice that if Ax has a strong ∇-model then for any formula $\varphi \in Form$, if φ strongly ∇-holds in this model then $\neg\varphi$ does not strongly ∇-hold in it, and so indeed Ax is consistent (for s-s-∇-consequence).

To see that for ∇-w-w-consequence a set of axioms may have a (weak) model and be inconsistent at the same time, consider $Ax = \{(\forall x)(p(x) \wedge \neg p(x))\}$. □

6. Sequents and signed formulas

In the rest of the paper we will present a formalized logic corresponding to the s-s-MK-consequence relation defined in the previous sections. In fact, we are going to develop two equivalent deduction systems for this consequence relation: a Gentzen-style sequent calculus and a natural deduction system.

Let us begin with a sequent calculus of Gentzen's type (see e.g. [Beth 59]). Recall that we are dealing with an arbitrary but fixed formalized logical language as presented in Section 2.

As usual for Gentzen-style calculi, we will formalize not just s-s-MK-consequence, but its generalization to the multi-conclusion case defined as follows: for any two sets of formulas $Ax, \Phi \subseteq Form$, we define

$$Ax \models_{ss}^{MK} \Phi$$

to mean that at least one formula $\varphi \in \Phi$ strongly MK-holds in every strong MK-model of Ax.

By a *sequent* we mean any pair $\langle \Gamma, \Delta \rangle$ of finite sets Γ and Δ of formulas, $\Gamma, \Delta \subset Form$. Sequents are traditionally written in the form $\Gamma \vdash \Delta$. Also traditionally, comma appearing in a sequent denotes the set-theoretic union; for example $\varphi, \Gamma \vdash \Delta$ should be read as $\{\varphi\} \cup \Gamma \vdash \Delta$.

Notice that since from now on we are dealing with only one consequence relation, we omit any decoration on the entailment symbol $\vdash$. Similarly, we omit any decoration on the semantic consequence and write simply $\models$ for $\models_{ss}^{MK}$, "satisfies" for "strongly MK-satisfies", "model" for "strong MK-model", etc.

A sequent $\Gamma \vdash \Delta$ is *valid* if for every model $\langle A, sta \rangle$ of Γ at least one formula $\delta \in \Delta$ holds in $\langle A, sta \rangle$.

Thus, for any (finite) set of formulas $Ax \subset Form$ and formula $\varphi \in Form$, the sequent $Ax \vdash \{\varphi\}$ is valid if and only if $Ax \models \varphi$. This implies that the syntactic entailment $\vdash$ in a valid sequent may indeed be viewed as a formal counterpart of (the generalization to the multi-conclusion case of) the semantic consequence relation we are studying.

The main problem we have to cope with to develop a formalized logic for the s-s-MK-consequence is, of course, that we are dealing with a three-valued calculus. One way of reducing this three-valued case to the standard situation is to introduce so-called signed formulas.

A *signed formula* has the form either $\mathsf{T}\varphi$ or $\mathsf{NT}\varphi$ (to be read "φ is true" and "φ is not true", respectively) where $\varphi \in Form$ is an arbitrary formula. Thus, the set of all signed formulas is defined

as follows:

$$SForm = \{\mathsf{T}\varphi, \mathsf{NT}\varphi \mid \varphi \in Form\}.$$

Let A be any logical structure with the semantics of formulas F_{MK}. The semantic function for signed formulas

$$SF\colon SForm \to State \to Bool$$

where $Bool = \{tt, ff\}$, is defined in a rather obvious way: for any formula $\varphi \in Form$ and state sta over A

$$SF.\mathsf{T}\varphi.sta = \begin{cases} tt & \text{if } F_{MK}.\varphi.sta = tt, \\ ff & \text{otherwise,} \end{cases}$$

and

$$SF.\mathsf{NT}\varphi.sta = \begin{cases} ff & \text{if } F_{MK}.\varphi.sta = tt, \\ tt & \text{otherwise.} \end{cases}$$

It is easy to see that we can always recover the value (in $Boolerr$) of any formula just by considering the truth of appropriate signed formulas:

Fact 6.1 For any formula $\varphi \in Form$, for any structure A and state sta over A:

1. $F_{MK}.\varphi.sta = tt$ iff $SF.\mathsf{T}\varphi.sta = tt$,
2. $F_{MK}.\varphi.sta = ff$ iff $SF.\mathsf{T}\neg\varphi.sta = tt$,
3. $F_{MK}.\varphi.sta = ee$ iff $SF.\mathsf{NT}\varphi.sta = SF.\mathsf{NT}\neg\varphi.sta = tt$.

□

Consider any structure A and state sta over A. Just as for formulas, for any signed formula $\rho \in SForm$ we say that A *satisfies* ρ *in* sta, or that ρ *holds in* $\langle A, sta\rangle$ if $SF.\rho.sta = tt$. For any set $\Omega \subseteq SForm$ of signed formulas, we say that $\langle A, sta\rangle$ is a *model of* Ω if each $\rho \in \Omega$ holds in $\langle A, sta\rangle$; we say that Ω is *satisfiable* if it has a model.

For any set $\Phi \subseteq Form$ we write $\mathsf{T}\Phi$ for $\{\mathsf{T}\varphi \mid \varphi \in \Phi\}$ and $\mathsf{NT}\Phi$ for $\{\mathsf{NT}\varphi \mid \varphi \in \Phi\}$.

Theorem 6.1 A sequent $\Gamma \vdash \Delta$ is valid if and only if the set $\mathsf{T}\Gamma \cup \mathsf{NT}\Delta$ of signed formulas is not satisfiable.

Proof

"$\Longrightarrow$": Suppose that $\mathsf{T}\Gamma \cup \mathsf{NT}\Delta$ is satisfiable, that is that it has a model $\langle A, sta\rangle$. Then, by our definitions, all formulas $\gamma \in \Gamma$ hold in $\langle A, sta\rangle$, but no $\delta \in \Delta$ does, and so the sequent $\Gamma \vdash \Delta$ is not valid.

"$\Longleftarrow$": Consider any model $\langle A, sta\rangle$ of Γ. Then, by our definitions, $\langle A, sta\rangle$ is a model of $\mathsf{T}\Gamma$. Assuming that $\mathsf{T}\Gamma \cup \mathsf{NT}\Delta$ is not satisfiable, it follows that for some $\delta \in \Delta$, $\mathsf{NT}\delta$ does not hold in $\langle A, sta\rangle$, i.e. δ holds in $\langle A, sta\rangle$. Thus, the sequent $\Gamma \vdash \Delta$ is valid.

□

7. Semantic tableaux

Theorem 6.1 ensures that we can consider (non-)satisfiability of sets of signed formulas instead of considering validity of sequents. A classical tool for verifying whether a set of signed formulas is satisfiable is the method of semantic tableaux introduced originally in [Beth 59] and then modified to cover the non-standard three-valued case in [Koletsos 76], where it was used to develop a sound and complete Gentzen-style formalization of the three-valued Kleene calculus.

In the sequel we follow the general line of [Koletsos 76], although we have a somewhat different set of rules, as we are dealing with a different calculus here. Therefore we omit most technicalities which may be found in [Koletsos 76] (this includes some technical lemmas and detailed proofs) and concentrate on explaining the general idea, highlighting the key points of the reasoning and discussing some details specific to the calculus we are formalizing.

Let us start with an informal description of the method.

A semantic tableau for a finite set Ω of signed formulas is a binary tree of signed formulas (to be more precise: of their occurrences) which may be used to determine whether Ω is satisfiable. It is built downwards (i.e. from the root towards the leaves) starting from a chain containing all the formulas in Ω. The basic idea is that each branch of the tree should contain all formulas which must be simultaneously satisfied if Ω is to be satisfied. Thus, each maximal branch represents an alternative "way" of satisfying Ω. A branch B is expanded downwards or split into two branches by appending new formula(s) which must hold in order that some formula $\rho \in$ B hold. If, for example, $\rho \cong \mathsf{T}(\alpha \wedge \beta)$, then B is expanded by appending $\mathsf{T}\alpha$ (and $\mathsf{T}\beta$) since the only way to make $\mathsf{T}(\alpha \wedge \beta)$ true is to make both $\mathsf{T}\alpha$ and $\mathsf{T}\beta$ true. If $\rho \cong \mathsf{NT}(\alpha \wedge \beta)$ then B is split at the end, with one subbranch containing $\mathsf{NT}\alpha$ and the other containing $\mathsf{NT}\beta$, since there are two ways to make $\mathsf{NT}(\alpha \wedge \beta)$ true: one is to make $\mathsf{NT}\alpha$ true, and the other is to make $\mathsf{NT}\beta$ true. The semantics of formulas may be used to determine a whole set of such rules for expanding and splitting branches. We refer to these rules as *tableau rules*. In fact, we have just mentioned three such tableau rules for the calculus we are formalizing:

$$\frac{\mathsf{T}(\alpha \wedge \beta)}{\mathsf{T}\alpha} \qquad \frac{\mathsf{T}(\alpha \wedge \beta)}{\mathsf{T}\beta} \qquad \frac{\mathsf{NT}(\alpha \wedge \beta)}{\mathsf{NT}\alpha \mid \mathsf{NT}\beta}$$

Each branch is expanded until we encounter a contradiction, i.e. discover a pair of signed formulas that cannot be simultaneously true. Then the branch is *closed* by applying one of the following two closure rules:

$$\frac{\mathsf{T}\varphi}{\mathsf{T}\neg\varphi\;\;\mathsf{T}\neg\varphi} \qquad \frac{\mathsf{T}\varphi}{\mathsf{NT}\varphi\;\;\mathsf{NT}\varphi}$$

which list the "forbidden" pairs of fomulas. Such a branch (traditionally marked by a double line at its end) is not expanded further, as it does not represent any possible way of satisfying Ω. The lack of what one might think to be a closure rule

$$\frac{\mathsf{NT}\varphi}{\mathsf{NT}\neg\varphi\;\;\mathsf{NT}\neg\varphi}$$

expresses the fact that the law of excluded middle does not hold for the logic we formalize.

With this ideas in mind let us try to give a bit more formal definition.

A *semantic tableau* for a set $\Omega = \{\rho_1, \ldots, \rho_n\}$ of signed formulas is a downward progressing tree of signed formulas which is either of the form

$$\begin{array}{c} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_n \end{array}$$

or has been obtained from another semantic tableau for Ω by applying one of the tableau rules given in Section 11, where a tableau rule of the form

$$\frac{\rho}{\sigma} \quad \text{or} \quad \frac{\rho}{\sigma_1 \mid \sigma_2}$$

may be applied to a branch B of a tableau if B contains ρ and then

- as a result of applying the rule $\frac{\rho}{\sigma}$ to B, this branch is expanded by attaching σ to its end, and
- as a result of applying the rule $\frac{\rho}{\sigma_1 \mid \sigma_2}$ to B, this branch is split at the end into two subbranches with σ_1 added at the end of the left subbranch and σ_2 added at the end of the right subbranch.

Some of the rules given in Section 11 deserve special attenion, as they reflect the specific pecularities of McCarthy's connectives. For example, the rule

$$\frac{\mathsf{T}(\alpha \vee \beta)}{\mathsf{T}\alpha \mid \mathsf{T}(\neg\alpha \wedge \beta)}$$

indicates the characteristic asymmetry of McCarthy's disjunction. Indeed, $\alpha \vee \beta$ is true if and only if either α is true (for the evaluation is "lazy") or $\neg\alpha \wedge \beta$ is true, i.e. α is false (but defined) and β is true.

By a dual analysis we obtain the rules:

$$\frac{\mathsf{NT}(\alpha \vee \beta)}{\mathsf{NT}\alpha} \quad \text{and} \quad \frac{\mathsf{NT}(\alpha \vee \beta)}{\mathsf{NT}\neg\alpha \mid \mathsf{NT}\beta}$$

Indeed, if $\alpha \vee \beta$ is not true then certainly α cannot be true, and moreover, either $\neg\alpha$ or β must not be true.

Since we use Kleene's quantifiers, the rules for quantified formulas are the same as in [Koletsos 76] and we do not discuss them here.

It should be pointed out, though, that similarly as in [Koletsos 76] for each connective (or quantifier) we have two sets of tableau rules: one for formulas formed using this connective, and another for negations of such formulas. This, of course, is a consequence of the fact that we are dealing with a three-valued calculus here, and hence we have to know the truth of both a formula and its negation to recover its original value (in the three-valued case, $\mathsf{T}\neg\varphi$ and $\mathsf{NT}\varphi$ are not equivalent — cf. Fact 6.1). Thus, the above rules describing disjunction must be considered together with the following:

$$\frac{\mathsf{T}\neg(\alpha \vee \beta)}{\mathsf{T}\neg\alpha} \qquad \frac{\mathsf{T}\neg(\alpha \vee \beta)}{\mathsf{T}\neg\beta} \qquad \frac{\mathsf{NT}\neg(\alpha \vee \beta)}{\mathsf{NT}\neg\alpha \mid \mathsf{NT}\neg\beta}$$

It is perhaps worth pointing out that alternatively we could have used here de Morgan's law in the form of the rules:

$$\frac{T\neg(\alpha \vee \beta)}{T(\neg\alpha \wedge \neg\beta)} \qquad \frac{NT\neg(\alpha \vee \beta)}{NT(\neg\alpha \wedge \neg\beta)}$$

We would need fewer tableaux rules then (as well as fewer rules in the formal logical systems we derive from them) but we feel that the overall justification of the soundness and completeness of the resulting systems would be less clear.

A branch of a semantic tableau is *closed* if it contains a pair of formulas appearing in one of the closure rules given in Section 11:

$$\frac{T\varphi}{\underline{\underline{T\neg\varphi}}} \qquad \frac{T\varphi}{\underline{\underline{NT\varphi}}}$$

A semantic tableau is *closed* if all its branches are closed.

A semantic tableau is *complete* if none of its open branches can be expanded any further by applying any of the rules.

Example 7.1 Consider a set $\Omega = \{T(\neg(\alpha \wedge \beta) \vee \gamma), NT(\alpha \vee \gamma)\}$ of signed formulas. We assume that α, β and γ are mutually distinct elementary formulas. A complete semantic tableau for Ω has the following form:

$$\begin{array}{l}
T(\neg(\alpha \wedge \beta) \vee \gamma) \\
NT(\alpha \vee \gamma) \\
\hline
\begin{array}{l|l|l}
T\neg(\alpha \wedge \beta) & & T(\neg\neg(\alpha \wedge \beta) \wedge \gamma) \\
\hline
T\neg\alpha & T(\alpha \wedge \neg\beta) & T\neg\neg(\alpha \wedge \beta) \\
NT\alpha & T\alpha & T\gamma \\
\hline
\underline{\underline{NT\neg\alpha}} \;|\; NT\gamma & T\neg\beta & T(\alpha \wedge \beta) \\
 & \underline{\underline{NT\alpha}} & T\alpha \\
 & & T\beta \\
 & & \underline{\underline{NT\alpha}}
\end{array}
\end{array}$$

The tableau is complete since we cannot expand its only open branch by applying any new rule, as all the rules applicable to the formulas of this branch have already been applied. Of course, the tableau is not closed — it has an open branch. □

Intuitively, a set of signed formulas with a closed semantic tableau cannot be satisfiable, as each of the alternative ways leading to its satisfiability results in a contradiction. The opposite implication is true as well:

Theorem 7.1 A finite set $\Omega \subset \mathit{SForm}$ of signed formulas is not satisfiable if and only if it has a closed semantic tableau.

Proof A detailed proof of the analogous result given in [Koletsos 76] may be applied also in our case with little modification reflecting the differences in the tableau rules (which in turn reflect the differences between *MK*- and *K*-calculi). Thus, we omit the details and just sketch the main ideas.

The "if" part seems intuitively obvious (see the remarks preceding the theorem) and can be proved by simple technical verification (induction on the structure of the closed semantic tableau).

The "only if" part is proved by constructing a model for every finite set of signed formulas that does not have a closed semantic tableau. Namely, for any such set Ω one can built a complete semantic tableau which has at least one open branch B. It turns out that the formulas of B form so-called *Hintikka set* of signed formulas (to put it briefly, B is closed under the tableau rules). This implies that B may be used to construct a model $\langle A, sta\rangle$ which satisfies all the formulas of B, and so is a model of Ω. The set of values of A may be taken to be the set of all terms of our language, with the semantics of function symbols defined in the obvious way. Then, for any $m = 1, 2, \ldots$, for $p \in Pred_m$ and terms $\vec{v} \in Value^m$, we define

$$P_m.p.\vec{v} = \begin{array}{ll} tt & \text{if } \mathsf{T}p(\vec{v}) \in \mathrm{B}, \\ ff & \text{if } \mathsf{T}\neg p(\vec{v}) \in \mathrm{B}, \\ ee & \text{otherwise.} \end{array}$$

Notice now that the elements of B with the derivability relation determined by the tableau rules form a graph with no infinite paths (even though B may be infinite). By an obvious induction on the structure of this graph we can show that any signed formula $\rho \in \mathrm{B}$ holds in $\langle A, sta\rangle$. □

Example 7.2 Theorem 7.1 implies that the set Ω of signed formulas considered in Example 7.1 is satisfiable. We can construct a model for it following the idea outlined in the proof of Theorem 7.1. The only open branch of the complete tableau for Ω given in Example 7.1 is $\mathrm{B} = \{\mathsf{T}(\neg(\alpha \wedge \beta) \vee \gamma), \mathsf{NT}(\alpha \vee \gamma), \mathsf{T}\neg(\alpha \wedge \beta), \mathsf{T}\neg\alpha, \mathsf{NT}\alpha, \mathsf{NT}\gamma\}$. Then the construction yields a structure A in which (sta is the identity):

$$\begin{array}{ll} F_{MK}.\alpha.sta = ff & \text{since } \mathsf{T}\neg\alpha \in \mathrm{B} \\ F_{MK}.\beta.sta = ee & \text{since neither } \mathsf{T}\beta \in \mathrm{B} \text{ nor } \mathsf{T}\neg\beta \in \mathrm{B} \\ F_{MK}.\gamma.sta = ee & \text{since neither } \mathsf{T}\gamma \in \mathrm{B} \text{ nor } \mathsf{T}\neg\gamma \in \mathrm{B} \end{array}$$

Hence:

$$\begin{array}{ll} F_{MK}.(\neg(\alpha \wedge \beta) \vee \gamma).sta = tt, & \text{and so } SF.\mathsf{T}(\neg(\alpha \wedge \beta) \vee \gamma).sta = tt, \\ F_{MK}.(\alpha \vee \gamma).sta = ee, & \text{and so } SF.\mathsf{NT}(\alpha \vee \gamma).sta = tt. \end{array}$$

Thus indeed, $\langle A, sta\rangle$ is a model of Ω. □

Theorems 6.1 and 7.1 directly imply the following key fact.

Corollary 7.1 A sequent $\Gamma \vdash \Delta$ is valid if and only if the set $\mathsf{T}\Gamma \cup \mathsf{NT}\Delta$ of signed formulas has a closed tableau. □

8. A complete deduction system for valid sequents

Corollary 7.1 hints at a certain method of obtaining a deduction system for valid sequents out of the set of tableau rules. This method was discussed in [Koletsos 76], although we find the presentation there far from clear. Moreover, that paper is not easily available, and since we certainly feel that the method is worth popularising, we will outline it in this section, giving in addition explicit forms of inference rules for the sequent calculus corresponding to the basic types of tableau rules (in [Koletsos 76] each inference rule was derived separately, which obscured the general pattern of reasoning).

First some notation: for any finite set $\Omega \subset \mathit{SForm}$ of signed formulas, by $\mathit{seq}.\Omega$ we denote the sequent $\Gamma \vdash \Delta$ where $\Omega = \mathsf{T}\Gamma \cup \mathsf{NT}\Delta$. Notice that Corollary 7.1 may be rephrased as follows: for any finite set $\Omega \subset \mathit{SForm}$ of signed formulas, $\mathit{seq}.\Omega$ is valid if and only if Ω has a closed tableau.

Now, consider a tableau rule of the form $\dfrac{\rho}{\sigma}$.

Then, by the definitions of Section 7, any branch of a semantic tableau that contains ρ may be expanded by adding σ to it. Consequently, for any finite set $\Omega \subset \mathit{SForm}$ of signed formulas, $\Omega \cup \{\rho\}$ has a closed tableau provided that $\Omega \cup \{\rho, \sigma\}$ has a closed tableau. Hence, by Corollary 7.1, the sequent $\mathit{seq}.(\Omega \cup \{\rho\})$ is valid whenever the sequent $\mathit{seq}.(\Omega \cup \{\rho, \sigma\})$ is. For example, taking $\rho \cong \mathsf{T}\varphi$ and $\sigma \cong \mathsf{T}\psi$, the tableau rule of the form

$$\frac{\mathsf{T}\varphi}{\mathsf{T}\psi}$$

gives rise to the following inference rule for deriving valid sequents:

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}$$

In other words, if we have a tableau rule $\dfrac{\mathsf{T}\varphi}{\mathsf{T}\psi}$ then the inference rule $\dfrac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}$ leads from valid sequents to valid sequents.

In particular, since one of our tableau rules is $\dfrac{\mathsf{T}(\alpha \wedge \beta)}{\mathsf{T}\alpha}$, we can include in the deduction system for valid sequents the following inference rule:

$$\frac{\Gamma, \alpha \wedge \beta, \alpha \vdash \Delta}{\Gamma, \alpha \wedge \beta \vdash \Delta}$$

The situation with the splitting rules of the form $\dfrac{\rho}{\sigma_1 \mid \sigma_2}$ is quite similar. Here we know that for any finite set $\Omega \subset \mathit{SForm}$ of signed formulas, $\Omega \cup \{\rho\}$ has a closed tableau provided that both $\Omega \cup \{\rho, \sigma_1\}$ and $\Omega \cup \{\rho, \sigma_2\}$ have closed tableaux. Hence, the sequent $\mathit{seq}.(\Omega \cup \{\rho\})$ is valid whenever the sequents $\mathit{seq}.(\Omega \cup \{\rho, \sigma\})$ and $\mathit{seq}.(\Omega \cup \{\rho, \sigma_1\})$ are. For example, taking $\rho \cong \mathsf{NT}\varphi$, $\sigma_1 \cong \mathsf{NT}\psi_1$ and $\sigma_2 \cong \mathsf{NT}\psi_2$, the tableau rule

$$\frac{\mathsf{NT}\varphi}{\mathsf{NT}\psi_1 \mid \mathsf{NT}\psi_2}$$

gives rise to the following inference rule for deriving valid sequents:

$$\frac{\Gamma \vdash \Delta, \varphi, \psi_1 \quad \Gamma \vdash \Delta, \varphi, \psi_2}{\Gamma \vdash \Delta, \varphi}$$

In particular, since one of our tableau rules is $\dfrac{\mathsf{NT}(\alpha \wedge \beta)}{\mathsf{NT}\alpha \mid \mathsf{NT}\beta}$, we can include in the deduction system for valid sequents the following inference rule:

$$\frac{\Gamma \vdash \Delta, \alpha \wedge \beta, \alpha \quad \Gamma \vdash \Delta, \alpha \wedge \beta, \beta}{\Gamma \vdash \Delta, \alpha \wedge \beta}$$

What is not very elegant here is that apparently we are forced to include φ and $\alpha \wedge \beta$, respectively, in the top sequents of the above rules, although intuitively they should not be necessary. In fact, we can get rid of them in the presence of the universally accepted *thinning rules*:

$$\frac{\Gamma \vdash \Delta}{\Gamma, \varphi \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi}$$

which clearly lead from valid sequents to valid sequents. We omit a simple proof that in the presence of the thinning rules the four rules of we have mentioned above are equivalent to, respectively:

$$\frac{\Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, \psi_1 \quad \Gamma \vdash \Delta, \psi_2}{\Gamma \vdash \Delta, \varphi}$$

and

$$\frac{\Gamma, \alpha \vdash \Delta}{\Gamma, \alpha \wedge \beta \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, \alpha \quad \Gamma \vdash \Delta, \beta}{\Gamma \vdash \Delta, \alpha \wedge \beta}$$

The above considerations are summarized in the following table which gives the inference rules for deriving valid sequents induced by tableau rules of the form we are using:

Tableau rule	Inference rule for sequents
$\dfrac{\mathsf{T}\varphi}{\mathsf{T}\psi}$	$\dfrac{\Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}$
$\dfrac{\mathsf{NT}\varphi}{\mathsf{NT}\psi}$	$\dfrac{\Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi}$
$\dfrac{\mathsf{T}\varphi}{\mathsf{T}\psi_1 \mid \mathsf{T}\psi_2}$	$\dfrac{\Gamma, \psi_1 \vdash \Delta \quad \Gamma, \psi_2 \vdash \Delta}{\Gamma, \varphi \vdash \Delta}$
$\dfrac{\mathsf{NT}\varphi}{\mathsf{NT}\psi_1 \mid \mathsf{NT}\psi_2}$	$\dfrac{\Gamma \vdash \Delta, \psi_1 \quad \Gamma \vdash \Delta, \psi_2}{\Gamma \vdash \Delta, \varphi}$

It seems even more obvious how the closure rules for tableaux give rise to axioms of the sequent calculus. By the closure rules, for any finite set $\Omega \subset \mathit{SForm}$ of signed formulas, the sets $\Omega \cup \{\mathsf{T}\varphi, \mathsf{T}\neg\varphi\}$ and $\Omega \cup \{\mathsf{T}\varphi, \mathsf{NT}\varphi\}$ have closed tableaux, and so the corresponding sequents:

$$\Gamma, \varphi, \neg\varphi \vdash \Delta \quad \text{and} \quad \Gamma, \varphi \vdash \Delta, \varphi$$

may be taken as axioms of the calculus for deriving valid sequents.

Applying the above schema to each of the tableau rules (given in Section 11, the first column) we obtain a set of inference rules for deriving valid sequents induced by the tableau rules. These inference rules are listed in Section 11 (second column) together with the extra thinning rules and the axioms.

We have obtained thus a deductive system for our sequent calculus. From now on it will be denoted by $\mathcal{SC}$. For any sequent $\Gamma \vdash \Delta$, we will write $\Gamma \vdash_{\mathcal{SC}} \Delta$ if the sequent $\Gamma \vdash \Delta$ is derivable in $\mathcal{SC}$, i.e. if it can be derived from the axioms of $\mathcal{SC}$ by means of the rules of $\mathcal{SC}$ (applied finitely many times).

Theorem 8.1 A sequent $\Gamma \vdash \Delta$ is valid if and only if it is derivable in $\mathcal{SC}$, that is, $\Gamma \models \Delta$ if and only if $\Gamma \vdash_{\mathcal{SC}} \Delta$.

Proof By Corollary 7.1, it is enough to prove that for any finite set $\Omega \subset \mathit{SForm}$ of signed formulas, Ω has a closed tableau if and only if the sequent $\mathit{seq}.\Omega$ is derivable in $\mathcal{SC}$.

A proof of the "if" part (soundness of $\mathcal{SC}$) has been in fact outlined above. We have defined the inference rules and axioms of $\mathcal{SC}$ in such a way that any derivation of a sequent $\Gamma \vdash_{\mathcal{SC}} \Delta$ determines a closed tableau for the set $\mathbb{T}\Gamma \cup \mathbb{N}\mathbb{T}\Delta$ (we omit a formal proof by induction on the structure of the derivation).

And vice versa: any closed tableau for a set Ω determines a derivation of the sequent $\mathit{seq}.\Omega$ in $\mathcal{SC}$. This, however, is exactly the "only if" part of the theorem (completeness of $\mathcal{SC}$). Again, we just outline a formal proof. We proceed by induction on the structure of the closed tableau, moving from the leaves of the tableau towards its root.

For any node in the semantic tableau, which we will identify with the branch B leading from the root to this node, we prove that the sequent corresponding to the set of signed formulas on this branch (we write $\mathit{seq}.\mathrm{B}$ to denote this sequent) is derivable in $\mathcal{SC}$. Since all the branches of the tableau are closed, this clearly holds for the leaves of the tableau (the corresponding sequents are axioms of $\mathcal{SC}$). Then, consider any branch B and suppose that in the semantic tableau a rule of the form $\frac{\rho}{\sigma}$ is applied to B. By the inductive hypothesis we know that the sequent corresponding to the branch resulting from B by the application of this rule is derivable in $\mathcal{SC}$. However, this sequent is just $\mathit{seq}.(\mathrm{B} \cup \{\sigma\})$ and the inference

$$\frac{\mathit{seq}.(\mathrm{B} \cup \{\sigma\})}{\mathit{seq}.\mathrm{B}}$$

is an instance of the inference rule of $\mathcal{SC}$ corresponding to the above tableau rule (recall that $\rho \in \mathrm{B}$ since the above tableau rule was applied to B). Hence, the sequent $\mathit{seq}.\mathrm{B}$ is indeed derivable in $\mathcal{SC}$. The case in which a splitting rule is applied to B may be dealt with in the same way.

We stop the induction at the node for which B is just the initial set Ω of signed formulas. Recall that since the tableau is closed, it is finite, and so the induction indeed terminates. A derivation of the sequent $\mathit{seq}.\Omega$ in $\mathcal{SC}$ may be recovered from the tableau as the induction steps indicate. □

9. A natural deduction system

Intuitively, by a natural deduction we mean the process of deducing conclusions from assumptions used in everyday life. Such a process is formalized by two types of logical systems: Gentzen-style sequent calculi (like that discussed in Section 8) and so-called *natural deduction systems*. An extensive classical study of natural deduction systems may be found in [Prawitz 65]. Here we will only give a brief description of such systems and develop such a system for the logic we are studying.

A natural deduction system is composed solely of inference rules, used for drawing immediate conclusions from given assumptions. The inference rules are of the form

$$\frac{\begin{array}{ccc}(\Gamma_1) & & (\Gamma_n)\\ \varphi_1 & \cdots & \varphi_n\end{array}}{\varphi}$$

where $\varphi_1, \ldots, \varphi_n$ and φ are formulas and $\Gamma_1, \ldots, \Gamma_n$ are finite sets of formulas. Such a rule should be read: if we have derived $\varphi_1, \ldots, \varphi_n$ from $\Gamma_1, \ldots, \Gamma_n$, respectively, then φ is true without any

additional assumptions, i.e. we can derive φ from $\varphi_1, \ldots, \varphi_n$ "discharging" assumptions $\Gamma_1, \ldots, \Gamma_n$ used to derive $\varphi_1, \ldots, \varphi_n$, respectively.

A typical simple form of such a rule is, of course, when all the sets $\Gamma_1, \ldots, \Gamma_n$ are empty. We write such rules simply as follows:

$$\frac{\varphi_1 \ \ldots \ \varphi_n}{\varphi}$$

which should be read: if all $\varphi_1, \ldots, \varphi_n$ are true then φ is true as well.[2]

A standard example of a rule where some assumptions are indeed discharged is the following rule of classical logic

$$\frac{\alpha \vee \beta \quad \begin{matrix}(\alpha)\\ \varphi\end{matrix} \quad \begin{matrix}(\beta)\\ \varphi\end{matrix}}{\varphi}$$

which should be read: if $\alpha \vee \beta$ is true, we have derived φ from α, and we have derived φ from β, then φ is true without any additional assumptions (that is, we can "discharge" assumptions α and β previously used to prove φ).

The notion of a *derivation tree* in such a natural deduction system is defined inductively. Any (occurrence of a) formula is such a tree. More complex trees are formed using the inference rules of the system in the obvious way. For example, if we have an inference rule $\frac{\varphi_1 \ \ldots \ \varphi_n}{\varphi}$ and we have built derivation trees $T_1, \ldots, T_n$ with formulas $\varphi_1, \ldots, \varphi_n$, respectively, at their roots, then we can use this rule to form a derivation tree with the formula φ at its root and with the trees $T_1, \ldots, T_n$ as the immediate subtrees.

The case of a rule like

$$\frac{\alpha \vee \beta \quad \begin{matrix}(\alpha)\\ \varphi\end{matrix} \quad \begin{matrix}(\beta)\\ \varphi\end{matrix}}{\varphi}$$

is a bit more complex. Namely, given derivation trees T_1, T_2 and T_3 with formulas $\alpha \vee \beta$, φ, and φ, respectively, at their roots, we can use this rule to form a new derivation tree with the formula φ at its root and T_1, T_2 and T_3 as its immediate subtrees (as before) but additionally we are allowed to mark occurrences of the formula α at the leaves of T_2 and occurrences of β at the leaves of T_3 as "discharged".

In general, given an inference rule

$$\frac{\begin{matrix}(\Gamma_1)\\ \varphi_1\end{matrix} \quad \ldots \quad \begin{matrix}(\Gamma_n)\\ \varphi_n\end{matrix}}{\varphi}$$

and derivation trees $T_1, \ldots, T_n$ with formulas $\varphi_1, \ldots, \varphi_n$, respectively, at their roots, we can use this rule to form a derivation tree with the formula φ at its root, with the trees $T_1, \ldots, T_n$ as its immediate subtrees, and with the occurrences of formulas in $\Gamma_1, \ldots, \Gamma_n$ at the leaves of the trees $T_1, \ldots, T_n$, respectively, marked as "discharged".

[2] Natural deduction systems with all inference rules of this form are sometimes referred to as *Hilbert-style systems*. In fact, Gentzen-style sequent calculi may be viewed as Hilbert-style systems over a formal language where sequents are viewed as individual formulas.

Now, a derivation of a formula φ from assumptions Γ in a natural deduction system is any derivation tree in this system with the formula φ at its root and all the formulas at its leaves that are not marked as "discharged" in the set Γ.

There is an obvious way to compare natural deduction systems with Gentzen-style systems (over the same logical language). For example, we say that a Gentzen-style system is equivalent to a natural deduction system whenever a sequent of the form $\Gamma \vdash \{\delta\}$ is derivable in the Gentzen-style system if and only if there is a derivation of δ from Γ in the natural deduction system.

Notice that we have to consider here again only single-conclusion sequents, as there is no obvious way of generalizing natural deduction systems to the multi-conclusion case. From now on, single-conclusion sequents as above will be written in the form $\Gamma \vdash \delta$. In this context it is important to realize that, roughly, in the Gentzen-style sequent calculus we have defined in the previous section a single-conclusion sequent may be derived if and only if it may be derived using only single-conclusion sequents.

In the rest of this section we will develop a natural deduction system *NDS* equivalent to the sequent calculus for deriving valid sequents, *SC*, defined in the previous section.

There can be no uniform way to define a natural deduction system equivalent to a given Gentzen-style sequent calculus. There is, however, an obvious strategy which we will try to employ here. Roughly:

- for each axiom $\Gamma \vdash \delta$ of *SC*, there must be a rule of *NDS* allowing us to derive δ from Γ;
- for each inference rule of *SC* of the form

$$\frac{\Gamma_1 \vdash \delta_1 \quad \ldots \quad \Gamma_n \vdash \delta_n}{\Gamma \vdash \delta}$$

  there must be a rule of *NDS* allowing us to construct a derivation of δ from Γ given derivations of $\delta_1, \ldots, \delta_n$ from, respectively, $\Gamma_1, \ldots, \Gamma_n$.

There is no uniform recipe for finding an *NDS* rule corresponding in the above sense to a given *SC* rule. We must simply examine the rules of *SC* (in their single-conclusion versions) and guess an appropriate *NDS* rule. We will discuss a few typical cases.

The simplest case is that of an inference rule of *SC* of the form

$$\frac{\Gamma \vdash \varphi_1 \quad \ldots \quad \Gamma \vdash \varphi_n}{\Gamma \vdash \psi}$$

which clearly corresponds to a natural deduction inference rule of the form

$$\frac{\varphi_1 \quad \cdots \quad \varphi_n}{\psi}$$

Then, an *SC* inference rule of the form

$$\frac{\Gamma, \varphi \vdash \delta}{\Gamma, \psi \vdash \delta}$$

induces in the above sense an *NDS* rule $\frac{\psi}{\varphi}$. Indeed, given a derivation of δ from $\Gamma \cup \{\varphi\}$ we can replace the leave(s) holding the formula φ by derivation(s) of φ from ψ built in the trivial way using this rule and then we obtain a derivation of δ from $\Gamma \cup \{\psi\}$.

The most complicated case is that of the *SC* rules like

$$\frac{\Gamma, \alpha \vdash \delta \quad \Gamma, \neg\alpha \wedge \beta \vdash \delta}{\Gamma, \alpha \vee \beta \vdash \delta} \qquad \frac{\Gamma, \neg\alpha \vdash \delta \quad \Gamma, \alpha \wedge \neg\beta \vdash \delta}{\Gamma, \neg(\alpha \wedge \beta) \vdash \delta}$$

Then, the corresponding natural deduction rules are:

$$\frac{\begin{array}{ccc} & (\alpha) & (\neg\alpha \wedge \beta) \\ \alpha \vee \beta & \delta & \delta \end{array}}{\delta} \qquad \frac{\begin{array}{ccc} & (\neg\alpha) & (\alpha \wedge \neg\beta) \\ \neg(\alpha \wedge \beta) & \delta & \delta \end{array}}{\delta}$$

It is clear how the axioms of *SC* translate to the rules of *NDS*: the axiom $\Gamma, \neg\varphi, \varphi \vdash \delta$ induces the rule $\frac{\varphi \quad \neg\varphi}{\delta}$. The other axiom, $\Gamma, \varphi \vdash \varphi$ needs no natural deduction rule, since by definition there is always a derivation of φ from any set that contains φ.

It remains to consider the thinning rules. The rule $\frac{\Gamma \vdash \delta}{\Gamma, \varphi \vdash \delta}$ needs no natural deduction rule, as any derivation of δ from Γ is a derivation of δ from any set that includes Γ.

The other thinning rule, $\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi}$, deserves some special attention (although it does not require any natural deduction rule either) as it is the only rule that allows us to derive a single-conclusion sequent from a non-single-conclusion one. This is the case if we take $\Delta = \emptyset$. However, it is quite clear then that the only way to derive the sequent $\Gamma \vdash \emptyset$ is using the axiom $\Gamma', \neg\varphi, \varphi \vdash \Delta$, and then we can apply the rule $\frac{\varphi \quad \neg\varphi}{\delta}$ to obtain a derivation of δ from Γ.

We can apply the above ideas to all the rules of the *SC* calculus listed in Section 11 (second column) to obtain the set of natural deduction rules listed in Section 11, third column. We will refer to this natural deduction system as *NDS*, and write $\Gamma \vdash_{NDS} \delta$ if there is a derivation of δ from Γ in *NDS*.

Theorem 9.1 If a sequent $\Gamma \vdash \delta$ is derivable in *SC* then there is a derivation of δ from Γ, that is,

$$\Gamma \vdash_{SC} \delta \qquad \text{implies} \qquad \Gamma \vdash_{NDS} \delta$$

Proof Follows directly by induction on the derivation of $\Gamma \vdash \delta$ in *SC*; the induction steps were informally described above. □

Theorem 9.2 For any finite set $\Gamma \subset Form$ of formulas and formula $\delta \in Form$,

$$\Gamma \models \delta \qquad \text{iff} \qquad \Gamma \vdash_{NDS} \delta$$

Proof The "if" part (soundness of *NDS*) may be checked directly by examining each of the inference rules of *NDS*.

The "only if" part (completeness of *NDS*) follows directly from Theorems 8.1 and 9.1. □

10. Final remarks

To conclude the paper, let us try to indicate briefly how the logical system presented here may be used in program specification and verification. Very roughly, we have the following model in mind:

Most programming languages used nowadays are parameterized by an underlying structure consisting of data the programs of the language operate on and of functions and predicates the programs may use as elementary operations and predicates on the data. Given such a structure, the (semantics of) programs are usually presented in terms of their input/output behaviour as state transformations. Three-valued predicates over this structure may be defined in our formalized logic by (open) formulas. Now, in our view, they may be used to describe adequately properties of states. Consequently, properties of state transformations (i.e., properties of programs) may be formulated in terms of so-called *superpredicates*, i.e. relations between predicates. This view has been developed and presented in detail in [Blikle 81a, 81b] where four such superpredicates were discussed (cf. also [Blikle 88]). Since they are all mutually definable in terms of each other, let us concentrate on one of them.

Let A be a structure and let p, q: *State* $\to$ *Boolerr* be two predicates over A. We say that p *is stronger than* q, written $p \Rightarrow q$, if $q.sta = tt$ whenever $p.sta = tt$ for every state sta over A. The superpredicate

$$\Rightarrow : Predicate \times Predicate \to \{tt, ff\}$$

is not expressible in our formalized language: roughly, there is no formula $st(\_,\_)$ with two "holes" such that for any structure A (with the semantics of formulas F_∇, $\nabla \in \{MK, K\}$) and any two formulas $\varphi, \psi \in Form$, $F_\nabla.\varphi \Rightarrow F_\nabla.\psi$ if and only if the formula $st(\varphi, \psi)$ strongly (resp., weakly) holds in A (cf. [Blikle 87]). It is easy to see that to build such a formula we need a non-monotone propositional connective. For example, if the definedness operator Δ (cf. Sec. 2) were added to our formalized language then

$$st(\varphi, \psi) \cong \Delta\varphi \supset (\varphi \supset (\Delta\psi \wedge \psi))$$

There is, however, another possibility which is more in the spirit of the system we present here. Namely, we can use the s-s-∇-consequence relation (which itself may be viewed as a non-monotone superpredicate):

Let us assume that the class of data structures on which the considered programs are expected to operate is definable by a set Ax of closed formulas. In fact, this in turn may be viewed as a specification of programs that implement the "primitive" functions and predicates of the data structure. Then, for any two formulas $\varphi, \psi \in Form$, for any structure A (with the semantics of formulas F_∇, $\nabla \in \{MK, K\}$) in this class (i.e., strongly satisfying Ax)

$$F_\nabla.\varphi \Rightarrow F_\nabla.\psi \quad \text{iff} \quad Ax, \varphi \models^{\nabla}_{ss} \psi$$

Of course, the latter fact may be proved using the formal logics we have presented here (for $\nabla = MK$).

It should be emphasized, however, that yet another approach is possible. Superpredicates are total, yielding always true or false, and so they may be viewed as classical predicates of a classical second-order theory we are working in. Thus, this classical second-order theory may be used directly to prove relationships between partial predicates, and hence properties of programs as well. In this approach, suggested in [Blikle 88], the need for a special three-valued logic disappears at all.

Much more experience is necessary to decide which of the approaches outlined above is best suited for practical applications.

11. Appendix: The three formal systems

Tableau rules	Sequent calculus $\mathcal{SC}$	Natural deduction system *NDS*
$\dfrac{T(\alpha \wedge \beta)}{T\alpha}$	$\dfrac{\Gamma, \alpha \vdash \Delta}{\Gamma, \alpha \wedge \beta \vdash \Delta}$	$\dfrac{\alpha \wedge \beta}{\alpha}$
$\dfrac{T(\alpha \wedge \beta)}{T\beta}$	$\dfrac{\Gamma, \beta \vdash \Delta}{\Gamma, \alpha \wedge \beta \vdash \Delta}$	$\dfrac{\alpha \wedge \beta}{\beta}$
$\dfrac{NT(\alpha \wedge \beta)}{NT\alpha \mid NT\beta}$	$\dfrac{\Gamma \vdash \Delta, \alpha \quad \Gamma \vdash \Delta, \beta}{\Gamma \vdash \Delta, \alpha \wedge \beta}$	$\dfrac{\alpha \quad \beta}{\alpha \wedge \beta}$
$\dfrac{T\neg(\alpha \wedge \beta)}{T\neg\alpha \mid T(\alpha \wedge \neg\beta)}$	$\dfrac{\Gamma, \neg\alpha \vdash \Delta \quad \Gamma, \alpha \wedge \neg\beta \vdash \Delta}{\Gamma, \neg(\alpha \wedge \beta) \vdash \Delta}$	$\dfrac{\begin{matrix}(\neg\alpha) & (\alpha \wedge \neg\beta) \\ \delta & \delta\end{matrix}}{\neg(\alpha \wedge \beta) \quad \delta}$
$\dfrac{NT\neg(\alpha \wedge \beta)}{NT\neg\alpha}$	$\dfrac{\Gamma \vdash \Delta, \neg\alpha}{\Gamma \vdash \Delta, \neg(\alpha \wedge \beta)}$	$\dfrac{\neg\alpha}{\neg(\alpha \wedge \beta)}$
$\dfrac{NT\neg(\alpha \wedge \beta)}{NT\alpha \mid NT\neg\beta}$	$\dfrac{\Gamma \vdash \Delta, \alpha \quad \Gamma \vdash \Delta, \neg\beta}{\Gamma \vdash \Delta, \neg(\alpha \wedge \beta)}$	$\dfrac{\alpha \quad \neg\beta}{\neg(\alpha \wedge \beta)}$
$\dfrac{T(\alpha \vee \beta)}{T\alpha \mid T(\neg\alpha \wedge \beta)}$	$\dfrac{\Gamma, \alpha \vdash \Delta \quad \Gamma, \neg\alpha \wedge \beta \vdash \Delta}{\Gamma, \alpha \vee \beta \vdash \Delta}$	$\dfrac{\begin{matrix}(\alpha) & (\neg\alpha \wedge \beta) \\ \delta & \delta\end{matrix}}{\alpha \vee \beta \quad \delta}$
$\dfrac{NT(\alpha \vee \beta)}{NT\alpha}$	$\dfrac{\Gamma \vdash \Delta, \alpha}{\Gamma \vdash \Delta, \alpha \vee \beta}$	$\dfrac{\alpha}{\alpha \vee \beta}$
$\dfrac{NT(\alpha \vee \beta)}{NT\neg\alpha \mid NT\beta}$	$\dfrac{\Gamma \vdash \Delta, \neg\alpha \quad \Gamma \vdash \Delta, \beta}{\Gamma \vdash \Delta, \alpha \vee \beta}$	$\dfrac{\neg\alpha \quad \beta}{\alpha \vee \beta}$
$\dfrac{T\neg(\alpha \vee \beta)}{T\neg\alpha}$	$\dfrac{\Gamma, \neg\alpha \vdash \Delta}{\Gamma, \neg(\alpha \vee \beta) \vdash \Delta}$	$\dfrac{\neg(\alpha \vee \beta)}{\neg\alpha}$
$\dfrac{T\neg(\alpha \vee \beta)}{T\neg\beta}$	$\dfrac{\Gamma, \neg\beta \vdash \Delta}{\Gamma, \neg(\alpha \vee \beta) \vdash \Delta}$	$\dfrac{\neg(\alpha \vee \beta)}{\neg\beta}$
$\dfrac{NT\neg(\alpha \vee \beta)}{NT\neg\alpha \mid NT\neg\beta}$	$\dfrac{\Gamma \vdash \Delta, \neg\alpha \quad \Gamma \vdash \Delta, \neg\beta}{\Gamma \vdash \Delta, \neg(\alpha \vee \beta)}$	$\dfrac{\neg\alpha \quad \neg\beta}{\neg(\alpha \vee \beta)}$
$\dfrac{T\neg\neg\alpha}{T\alpha}$	$\dfrac{\Gamma, \alpha \vdash \Delta}{\Gamma, \neg\neg\alpha \vdash \Delta}$	$\dfrac{\neg\neg\alpha}{\alpha}$
$\dfrac{NT\neg\neg\alpha}{NT\alpha}$	$\dfrac{\Gamma \vdash \Delta, \alpha}{\Gamma \vdash \Delta, \neg\neg\alpha}$	$\dfrac{\alpha}{\neg\neg\alpha}$

continued on the next page

Tableau rules	Sequent calculus $\mathcal{SC}$	Natural deduction system *NDS*
$\dfrac{\mathsf{T}((\forall x)\varphi[x])}{\mathsf{T}\varphi[t]}$ (*)	$\dfrac{\Gamma, \varphi[t] \vdash \Delta}{\Gamma, (\forall x)\varphi[x] \vdash \Delta}$ (*)	$\dfrac{(\forall x)\varphi[x]}{\varphi[t]}$ (*)
$\dfrac{\mathsf{NT}((\forall x)\varphi[x])}{\mathsf{NT}\varphi[y]}$ (**)	$\dfrac{\Gamma \vdash \Delta, \varphi[y]}{\Gamma \vdash \Delta, (\forall x)\varphi[x]}$ (**)	$\dfrac{\varphi[y]}{(\forall x)\varphi[x]}$ (**)
$\dfrac{\mathsf{T}\neg((\forall x)\varphi[x])}{\mathsf{T}\neg\varphi[y]}$ (**)	$\dfrac{\Gamma, \neg\varphi[y] \vdash \Delta}{\Gamma, \neg(\forall x)\varphi[x] \vdash \Delta}$ (**)	$\dfrac{\neg(\forall x)\varphi[x]}{\neg\varphi[y]}$ (**)
$\dfrac{\mathsf{NT}\neg((\forall x)\varphi[x])}{\mathsf{NT}\neg\varphi[t]}$ (*)	$\dfrac{\Gamma \vdash \Delta, \neg\varphi[t]}{\Gamma \vdash \Delta, \neg(\forall x)\varphi[x]}$ (*)	$\dfrac{\neg\varphi[t]}{\neg(\forall x)\varphi[x]}$ (*)
$\dfrac{\mathsf{T}((\exists x)\varphi[x])}{\mathsf{T}\varphi[y]}$ (**)	$\dfrac{\Gamma, \varphi[y] \vdash \Delta}{\Gamma, (\exists x)\varphi[x] \vdash \Delta}$ (**)	$\dfrac{(\exists x)\varphi[x]}{\varphi[y]}$ (**)
$\dfrac{\mathsf{NT}((\exists x)\varphi[x])}{\mathsf{NT}\varphi[t]}$ (*)	$\dfrac{\Gamma \vdash \Delta, \varphi[t]}{\Gamma \vdash \Delta, (\exists x)\varphi[x]}$ (*)	$\dfrac{\varphi[t]}{(\exists x)\varphi[x]}$ (*)
$\dfrac{\mathsf{T}\neg((\exists x)\varphi[x])}{\mathsf{T}\neg\varphi[t]}$ (*)	$\dfrac{\Gamma, \neg\varphi[t] \vdash \Delta}{\Gamma, \neg(\exists x)\varphi[x] \vdash \Delta}$ (*)	$\dfrac{\neg(\exists x)\varphi[x]}{\neg\varphi[t]}$ (*)
$\dfrac{\mathsf{NT}\neg((\exists x)\varphi[x])}{\mathsf{NT}\neg\varphi[y]}$ (**)	$\dfrac{\Gamma \vdash \Delta, \neg\varphi[y]}{\Gamma \vdash \Delta, \neg(\exists x)\varphi[x]}$ (**)	$\dfrac{\neg\varphi[y]}{\neg(\exists x)\varphi[x]}$ (**)
—	(thinning rules) $\dfrac{\Gamma \vdash \Delta}{\Gamma, \varphi \vdash \Delta}$ $\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi}$	—
Closure rules	**Axioms**	
$\dfrac{\mathsf{T}\varphi}{\mathsf{T}\neg\varphi}$	$\Gamma, \varphi, \neg\varphi \vdash \Delta$	$\dfrac{\varphi \quad \neg\varphi}{\delta}$
$\dfrac{\mathsf{T}\varphi}{\mathsf{NT}\varphi}$	$\Gamma, \varphi \vdash \Delta, \varphi$	—

(*) t is an arbitrary term;

(**) y is a new variable (i.e., respectively, y does not occur free in the formulas in the branch of the tableau where this rule is applied, y does not occur free in the formulas of Γ and of Δ, y does not occur free in non-discharged assumptions).

In the above table, α, β, φ and δ stand for arbitrary formulas, Γ and Δ for arbitrary finite sets of formulas, and x for an arbitrary identifier.

12. References

[Avron 87]
Simple consequence relations.
Report ECS-LFCS-87-30, Laboratory for Foundations of Computer Science, University of Edinburgh, June 1987.

[Barringer, Cheng & Jones 84]
Barringer, H., Cheng, J.H., Jones, C.B.
A logic covering undefinedness in program proofs.
Acta Informatica 21(1984), 251-269.

[Beth 59]
Beth, E.W.
The Foundations of Mathematics.
North-Holland 1959.

[Blikle 81a]
Blikle, A.
On the development of correct specified programs.
IEEE Transactions on Software Engineering SE-7(1981), 251-169.

[Blikle 81b]
Blikle, A.
The clean termination of iterative programs.
Acta Informatica 16(1981), 199-217.

[Blikle 87]
Blikle, A.
MetaSoft Primer: Towards a Metalanguage for Applied Denotational Semantics.
LNCS vol.288, Springer-Verlag 1987.

[Blikle 88]
Blikle, A.
A calculus of three-valued predicates for software specification and validation.
in: Proc. VDM-Europe Symposium 1988, LNCS, Springer-Verlag 1988, this volume.

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

[Goguen 77]
Goguen, J.A.
Abstract errors for abstract data types.
in: Proc. IFIP Working Conference on the Formal Description of Programming Concepts, St.Andrews 1977 (E.Neuhold, ed.), North-Holland 1978.

[Hoogewijs 79]
Hoogewijs, A.
On a formalization of the non-definedness notion.
Zeitschrift f. Math. Logik u. Grundlagen d. Math. 25(1979), 213-221.

[Hoogewijs 83]
Hoogewijs, A.
A partial predicate calculus in a two-valued logic.
Zeitschrift f. Math. Logik u. Grundlagen d. Math. 29(1983), 239-243.

[Hoogewijs 87]
Hoogewijs, A.
Partial-predicate logic in computer science.
Acta Informatica 24(1987), 381-393.

[Jones 86]
Jones, C.B.
Systematic Software Development Using VDM.
Prentice-Hall 1986.

[Jones 87]
Jones, C.B.
VDM proof obligations and their justification.
in: VDM – A Formal Method at Work, Proc. VDM-Europe Symposium 1987, LNCS vol.252
Springer-Verlag 1987, 260-286.

[Kleene 38]
Kleene, S.C.
On notation for ordinal numbers.
Journal of Symbolic Logic 3(1938), 150-155.

[Kleene 52]
Kleene, S.C.
Introduction to Mathematics.
North Holland 1952, then republished in 1957, 59, 62, 64, 71.

[Koletsos 76]
Koletsos, G.
Sequent calculus and partial logic.
MSc thesis, The University of Manchester 1976.

[McCarthy 61]
McCarthy, J.
A basis for a mathematical theory of computation.
Western Joint Computer Conference, May 1961; then published in: Computer Programming and Formal Systems (P.Braffort, D.Hirshberg, eds.) North-Holland 1967, 33-70.

[Owe 85]
Owe, O.
An approach to program reasoning based on a first-order logic for partial functions.
Research Report 89, Institute of Informatics, University of Oslo, February 1985.

[Prawitz 65]
Prawitz, D.
Natural Deduction.
Almquist & Wiksell, Stockholm 1965.