DEFINITIONS IN NONSTRICT POSITIVE FREE LOGIC*

RAYMOND D. GUMB

Computer Science Department
University of Massachusetts
Lowell, MA 01854, USA
email: gumb@cs.uml.edu

and

KAREL LAMBERT

Department of Philosophy
University ofCalifornia, Irvine
Irvine, CA 92717, USA
email: jlambert@benfranklin.hnet.uci.edu
and, after April 1st of each year: jlambert@ednet1.osl.or.gov

Abstract. Every "practical" programming language supplies the programmer with at least one nonstrict construct, such as the ALGOL60 arithmetic 'if-then-else' and the LISP 'cond'. Many programming languages also enable the user to define nonstrict functions. In some languages, this is accomplished through the lazy evaluation of procedure parameters, as realized, for example, by the call-by-name devices of ALGOL60 and SIMULA67 and the call-by-need mechanism of haskell. In other languages, such as Common LISP, a macro definition facility can serve a similar purpose. Programming languages that provide a mechanism for the user to define nonstrict functions are nonstrict languages, and we call the natural underlying logic of these languages *nonstrict positive free logic*. In this paper, we present the definition theory of nonstrict positive free logic. Suitable

---

* The groundwork for this paper was done while Gumb was on sabbatical at the University of California Irvine Philosophy Department visiting with Lambert. Gumb is grateful to Lambert and the UCI Philosophy Department, David Israel and the Stanford University Center for the Study of Language and Information, and Daniel Friedman and the Indiana University Computer Science Department for their hospitality during his sabbatical. Helpful comments have been made by Nino Cochiarella, William Farmer, Robert K. Meyer, John Peterson, Richmond Thomason, and Peter Woodruff.

transformations of sentences in standard logic into sentences in nonstrict positive free logic preserve many properties of definitions in standard logic.

Before embarking on details, here is one general piece of advice. One often hears that... some... logic is pointless because it can be translated into some simpler language in a first-order way. Take no notice of such arguments. There is no weight to the claim that the original system must therefore be replaced by the new one. What is essential is to single out important concepts and to investigate their properties. Dana Scott ([Scott 1970, 143])

1. The Motivation for Nonstrict Positive Free Logic. Free logic has been implicit in the literature on program specification and verification from the inception of the field.[1] Here, after Lambert and van Fraassen [Lambert & van Fraassen *1972*], we understand a free logic to be an extension of standard first-order logic in the sense that, when attention is restricted to closed formulas containing only variables as singular terms, the theorems and derivations of a free logic coincide with those of standard logic. Free logic is unlike standard logic in that singular terms need not have existential import.

Following Lambert [Lambert *1981*], one must distinguish negative free logic from positive free logic. In *negative free logic*, any atomic statement

---

[1] As noted in [Gumb *1996*], C. A. R. Hoare's treatment of the existential quantifier in his pioneering paper [Hoare *1969*] presupposes that the underlying logic is free. In an axiomatization of computer arithmetic, he states the law:

$$\neg \exists X (X = \max + 1)$$

where max is the largest integer representable. The assertion is unsatisfiable in standard logic, implying that the underlying logic is free. Similarly, the free interpretation of the quantifiers has roots in the early history of intuitionistic logic [Ruitenburg *1991*]. Attempts to formalize intuitionistic logic with the standard quantifiers has lead to a number of unsound versions of intuitionistic logic in the literature [Leblanc & Gumb *1983b*]. These difficulties can be attributed to the doctrine that the range of values of free variables (individual parameters), variables bound by the universal quantifier, and variables bound by the existential quantifier should all be the same. We follow Menger's counterpart of Occam's Razor, the *Law Against Miserliness*: "It is vain to do with less what requires more" [Menger *1979*, 106]. Conflation of the different senses of the term 'variable' are all too common in the literature, and it would be less error-prone, as well as better pedagogically, to use distinct terms for each of the senses, a point stressed, for example, in Thomason's introductory logic text [Thomason *1969*].

containing at least one singular term that does not refer to an existent[2] is false, whereas a *positive free logic* allows some such statement to be true. A related terminology is used in computer science.[3] An n-ary predicate p^n is *called strict in its i-th arguments* provided $p^n(t_1, \ldots, t_n)$ is false whenever t_i $(1 \leq i \leq n)$ does not refer to an existent,[4] and a function f^n is called *strict in its i-th arguments* provided $f^n(t_1, \ldots, t_n)$ does not refer to an existent if t_i does not. A predicate (function) is called *strict* if it is strict in all of its arguments. Combining the traditions in philosophical logic and computer science, it is natural to call a positive free logic that permits the definition of nonstrict predicates and functions a *nonstrict positive free logic.*[5]

Even if one were to grant that negative free logic is the more natural for formalizing all of conventional mathematics (which we do not!),[6] nonstrict positive free logic would still be needed to reason about nonstrict functions.

---

[2] In Kripke models of intuitionistic logic, the domain of the universal quantifier is, in effect, a superset of the domain of the existential quantifier. So, in free intuitionistic logic, a term that refers to no existent can be viewed as a term that refers to an object outside the domain of the existential quantifier. Quine's dictum "To be is to be the value of a bound" [Quine *1963*] must be modified to suit the intuitionistic case: "To be is to be the value of a variable bound by the existential quantifier", which appears uninformative. Similarly, the dictum requires modification to suit lazy programming languages. The variables in lambda abstractions in these languages (the formal parameters of functions) can be bound to an error value that is outside the domain of the existential quantifier.

[3] In computer science, perhaps the closest work to our own is that on the specification language COLD (Feijs & Jonkers *1992*], [Koymans & Renardel de Lavalette *1989*] and [Renardel de Lavalette *1994*]). However, the free logic underlying COLD is infinitary, and research on it has focused on strict predicates and functions. The design of COLD has been influenced by the work of Scott [Scott *1967*; *1979*].

[4] In this case, one might think of t_i as being undefined. There is another sense of "undefined" in computer science that means "not fully specified". Hoare [Hoare *1969*, 580] has argued that an advantage of "axiomatic semantics" for programming languages is that it can be used to leave aspects of a language undefined and allow implementation on differing hardware designs. In this connection, free logic might be used to prevent over specification by, for example, dropping axioms specifying the result of division by zero. We do not consider this approach further in the present paper.

[5] In contrast, Beeson's system [Beeson *1985*] and related logics such as those of Feferman [Feferman *1992*] Farmer [Farmer *1990*], and Parnas [Parnas *1993*] are naturally called *strict negative free logics.*

[6] If one were to grant this, our paper would be seen as bridging a gap between logic and computing practice that is specific to computer science. Gaps between logic and mathematical practice [Corcoran *1973*] are duplicated in gaps between logic and computing practice.

As we shall illustrate below, a positive free logic is needed to guarantee that sentences are assigned the correct truth-values. That a user can define nonstrict functions in a lazy programming language has deep implications for the definition theory of the underlying logic. In these languages, nonstrict positive free logic is required for formalizing user-defined functions.

Some classical mathematical theories are axiomatized appropriately in standard first-order logic, and, in order to treat these theories properly, results in standard logic must extend naturally to nonstrict positive free logic. For this reason, it is simpler to use an "inner domain, outer domain", two-valued semantics.[7]

Intuitively, "existents" populate the inner domain, while the outer domain has as its one and only inhabitant the nonexistent error object **err**. Programs compute the value of terms, but reasoning about programs determines the truth of sentences.[8]

In this paper, we present definition theory for a nonstrict positive free logic in which there is exactly one error object **err** to which all terms without existential import can refer.[9] By using a free logic, we are able to state the axioms of a mathematical theory without cluttering the axiomatization with error conditions, as would be required using restricted quantification in standard logic. For example, Peano's axiom:

$$\forall X \forall Y(\text{succ}\ (X) = \text{suc}(Y) \rightarrow\ X = Y)$$

can be stated in the usual way, whereas with restricted quantification, the axiom takes on the more awkward form:

$$\forall X :\ \text{Natno}\ \forall\ Y :\ \text{Natno}\ (\text{succ}\ (X) = \text{succ}\ (Y) \rightarrow\ X = Y)$$

or, translated into standard first-order logic, the even more verbose form:

$$\forall X \forall Y\ (\text{Natno(X)} \rightarrow (\text{Natno(Y)} \rightarrow (\text{succ}\ (X) = \text{suc}(Y) \rightarrow\ X = Y))).$$

Nonstrict positive free logic thus facilitates specifying computer systems and proving them correct.

---

[7] Other semantics such as supervaluations [Benccivenga *1986*] are available for positive free logics, but these do not lend themselves easily to extending classical results in definition theory to positive free logic.

[8] Functions and predicates in nonstrict positive free logics are treated in quite distinct manners. The Boolean "predicates" computed by a program are most naturally treated as boolean functions in the underlying free logic.

[9] Having exactly one error object identifies nontermination and all run-time errors. This is most natural in languages such as Miranda and haskell in which execution is aborted immediately when an error is raised. See [Thomason *1989*] for further discussion of this issue.

Nonstrict positive free logic enables the definition of nonstrict functions such as the constant 1-ary function 'zero' and the 3-ary function 'if-then-else'. The following equations hold for these functions:

$$\text{zero}(\text{err}) = 0$$

$$\text{if } 1 \text{ then } 2 \text{ else err} = 2$$

$$\text{if } 0 \text{ then } 2 \text{ else err} = \text{err}.$$

Similarly, a family of nonstrict n-ary projection functions are defined so that

$$\text{id}_i^n(x_1,\ldots,x_n) = x_i \quad (1 \leq i \leq n)$$

regardless of whether there is a j $(1 \leq j \leq n)$ such that $j_n = \text{err}$. Only nonstrict positive free logic can assign sentences such as these the correct truth-value, namely *true*. Nonstrict positive free logic is the first-order fragment of the natural underlying logic of lazy languages such as haskell and Miranda.

Nonstrict positive free logic is more expressive than standard logic if both are employed naturally. As well as facilitating reasoning about lazy programming languages, nonstrict positive free logic frequently can mechanically incorporate analogs of theorems that are adequately expressed in standard logic. In Section 5, we will present some transformations of sentences in standard logic into sentences in nonstrict positive free logic that preserve many properties of definitions in standard logic.

2. Nonstrict Positive Free Logic: Axiomatization and Semantics. Our nonstrict positive free logic, in many aspects, is similar to systems found in the literature, and so in our sketch of it is brief. Regarding the syntax, we follow the conventions of [Leblanc *1976*], except that we introduce function parameters (function symbols) and an individual constant 'err' that is intended to refer to the error object **err**. Specifically, we distinguish between *individual variables* (bound variables) $X, \ldots,$ written in upper case, and *individual parameters* (free variables) $a, \ldots,$ written in lower case. For each $n \geq 0$, we have countably many n-ary function parameters $f^n, \ldots$ [10] that can be used to construct complex singular terms. For the construction of sentences, we have sentence parameters $p^0, \ldots$ and n-ary predicate parameters $p^n, \ldots$. Writing $\equiv$ for syntactic identity, $A(a)$ for any sentence, and $A(X)$ for the result $A[X\backslash a]$ of substituting X for every

[10] In some contexts, for example in Section 3, we find it convenient to identify individual parameters with 0-ary function parameters.

occurrence of a in $A(a)$, the key clause in the inductive definition of the set of sentences is:

$$\forall X\, A(X) \text{ is a sentence if } A(a) \text{ is.}$$

A *literal* is an atomic sentence or the negation of one, and an *existential* is a sentence of the form $\exists X\,(X = t)$, where t is any singular term.

As *primitive logical constants,* we have the universal quantifier ($\forall$), negation ($\neg$), if-then ($\rightarrow$), the binary (nonstrict) equality predicate ($=$), and the error constant (err). Included amongst the nonprimitive nonlogical constants are the existential quantifier ($\exists$),and ($\wedge$), or ($\vee$), and if-and only-if ($\leftrightarrow$), which are defined in the usual way. As parenthesis omitting conventions, we understand that $\rightarrow$ associates to the right and all other binary operators associate to the left. The binding strength of the operators is given by the following order (where the operators are arranged in order of decreasing strength): the unary connectives, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$. The inequality predicate ($\neq$), the existence predicate (E!), the strict equality predicate ($\approx$), and the strict nonequality predicate ($\not\approx$), and are also logical constants, and are introduced below in a manner typical in the free logic literature. Our underlying free logic can be axiomatized much as Leblanc's axiomatization of the positive free logic $\mathbf{CQ}^{*}_{=}$ [Leblanc *1976*]:[11] We have a typical axiomatization of the propositional calculus, together with axioms for the universal quantifier:

$$\forall X(A \rightarrow B) \rightarrow \forall X\, A \rightarrow \forall X\, B$$

$$A \rightarrow \forall X\, A$$

$$\forall X\, A(X) \text{ if } A(a) \text{ is an axiom.}$$

$$\forall Y\, \exists X\, (X = Y)$$

$$\forall X\, A(X) \rightarrow \exists X\, A(X)^{12}$$

---

[11] In [Gumb *1989*], we use a Fitch-style natural deduction system.

[12] This axiom is not provable in $\mathbf{CQ}^{*}_{=}$. Semantically, it insures that there is at least one existent, as explained in [Bencivenga *1986*, 414]. For this reason, $\mathbf{CQ}^{*}_{=}$ is a *universally free logic* in the sense of [Meyer & Lambert *1968*].

and axioms for equality and the error constant. In the following, we understand $A[t'\backslash t]$ to be the sentence obtained by replacing every occurrence of t in A with t':

$$t = t$$

$$t = t' \rightarrow (A \rightarrow A[t\backslash t'])$$

$$t = t' \rightarrow (A \rightarrow A[t\backslash t'] \rightarrow A)$$

$$err = t \leftrightarrow \neg\, \exists X\, (X = t)$$

A distinctive mark of free logic is that the standard specification law

$$\forall X\, A(X) \rightarrow A(t)$$

is *not* provable, but the weaker specification law that is restricted to existents

$$\forall X\, A(X) \rightarrow \exists X\, (X = t) \rightarrow A(t)$$

is provable. In standard logic, $\exists X\, (X = t)$ is provable, whereas in free logic it need not be provable.

The following axioms could be used to introduce additional predicates that are logical constants peculiar to free logic:[13]

$$t \neq t' \leftrightarrow \neg\, (t = t')$$

$$E!(t) \leftrightarrow \exists X\, (X = t)$$

---

[13] These axioms are equivalent to the full explicit definitions given in Section 3. These predicates are nonprimitive logical constants because only logical constants occur in the definiens. An additional reason for considering them to be logical constants is that, in some axiomatizations of positive free logic with different sets of primitives, the Craig Interpolation Lemma goes through when they are counted as logical constants but fails otherwise. In the identity-free logic with E! as a primitive, for example, consider

$$\vdash \forall X\, A(X) \wedge \neg\, A(a) \rightarrow \neg\, E!(a)$$

The sentence $\neg\, E!(a)$serves as an interpolant, but no E!-free sentence can. Like equality, E! is indeed a predicate, but, unlike predicates like 'sleeps', it is a logical predicate.

$$t \approx t' \leftrightarrow \mathrm{E!}(t) \wedge t = t'$$

$$t \not\approx t' \leftrightarrow \mathrm{E!}(t) \wedge \mathrm{E!}(t') \wedge t \neq t'$$

The notion of a *proof* is understood in the usual manner. Understanding a *theory* T to be a set of sentences, we write $\vdash A$ if the sentence A is provable, and $T \vdash A$ as well as $\vdash_t A$ if the sentence A is provable from T.

Turning to the semantics, a model is a triple $M = (D_M, D_M, I_M)$, D_M is a nonempty set, $D_M = \mathbf{err}$, $\{\mathbf{err}\} \notin D_M$, and I_M is an interpretation function mapping symbols into appropriate objects. D_M is the *inner domain* of M, D_M is the *outer domain* of M, and **err** is the *error object*. When no ambiguity can arise, we shall omit subscripts.[14] The interpretation function I assigns objects of the appropriate types as follows:

1. I(err) = **err**,
2. $I(a) \in D \cup D$,
3. $I(f^n) \in (D \cup D)^n \rightarrow (D \cup D)$,[15]
4. $I(f^n(t_1, \ldots, t_n)) = I(f^n)(t_1), \ldots, I(f^n)(t_n)) \in D \cup D$,
5. $I(p^0) \in \{\mathbf{true}, \mathbf{false}\}$, and
6. $I(p^n) \subseteq (D \cup D)^n$.

Our inductive definition of truth is typical. We extend the domain of I to all sentences, understanding a sentence A to be **true**, written $I(A) = \mathit{true}$ or $M \models A$ provided:[16]

1. $I(t) = I(t')$ if $A \equiv t = t'$,

---

[14] Note that the predicate E! can be undecidable in a nonstrict positive free theory T in the sense that, for some term t, we may have neither $\vdash_T$ E! (t) nor $\vdash_T \neg$ E! (t). Consequently, free logic is not a fragment of "many-sorted" or "order-sorted" logic as those logics are normally understood. See, for example, [Schmidt-Strauss *1987*, 4]. Free logic is less cluttered than many-sorted or order sorted logic or logic with restricted quantification ([Hailperin *1957*]) as it is not burdened with a quantifier over the outer domain. Quantification over errors is of no mathematical interest and, because it is unneeded and verbose, it interferes with understanding system specifications and proofs of program correctness.

[15] We take $I(f^n)$ to be a total function on $D \cup D$, although the restriction of $I(f^n)$ to D need not be total. For example, if $I(\mathrm{E!}(g^1(t))) = \mathbf{false}$ for some term t, we force g to be total by insuring that $I((g(t)) = \mathbf{err}$.

[16] In the following, we use the sign $\equiv$ for syntactic identity, and we understand that an interpretation I' is an *a-variant* of an interpretation I provided $I'(b) = I(b)$ for every individual parameter $b \not\equiv a$.

2. $(I(t_1), \ldots, I(t_n)) \in I(p^n)$ if $A \equiv p^n(t_1, \ldots, t_n)$,
3. $I(B) =$ **false** if $A \equiv \neg B$,
4. $I(\mathrm{B}) =$ **false** or $I(C) =$ **true** if $A \equiv B \rightarrow C$, and
5. $I'(B(\mathrm{a})) =$ **true** for every a-variant I' of I such that $I'(a) \in \mathcal{D}$ if $A \equiv \forall X\, B(X)$, where a is the first individual parameter foreign to $\forall X\, B(X)$.

As usual, a sentence A is a *logical consequence* of a theory T, written $T \models A$ as well as $\models_T A$, provided that, in each model M such that $M \models B$ for every $B \in T$, we have $M \models A$. The sentence A is called *valid* if $\varnothing \models A$, where $\varnothing$ is the emptyset.

Nonstrict positive free is a reasonable logic, having intuitive Hilbert-style axiomatic, natural deduction, Gentzen-style sequent, and tableaux systems.[17] We note in particular that these deductive systems are strongly sound and complete and that the Deduction Theorem holds. Furthermore, it has a reasonable model theory because, as we shall discuss in more detail in Section 4, the Craig Interpolation Lemma and hence the Beth Definability Theorem hold.[18]

---

[17] For further information, see [Leblanc *1976*] and [Bencivenga *1986*] on axiomatizations, [Lambert & van Fraassen *1972*], [Bencivenga *1986*], and [Gumb 1989] on natural deduction, [Bencivenga *1986*] on sequents, and [Gumb *1979b*] and [Gumb *1984*] on tableauxs. All of these systems with the exception of that in [Gumb *1989*] require minor modifications to accommodate the unique error object **err**. We sketch the modifications required for tableaux in Section 4.

[18] A logic underlying a specification language for computer systems must be reasonable, even if formal verification of complete programs is never attempted. For example, it must be feasible to prove the equivalence of specifications in order to establish that program transformations, as used in compiler optimizations, preserve meaning. Tentative notes on desirable properties of specification languages, can be found in [Gumb *1982*]. Corcoran [Corcoran *1980*] argues that even classical mathematicians are not interested in logics solely for their model-theoretic prowess or, in our terms, their capacity to provide semantic "specifications". Corcoran studies "slightly augmented first-order logic" that has one predicate variable. The variable, when it occurs in a sentence, is given the generality interpretation. Corcoran argues that slightly augmented first-order logic is of little mathematical interest even though, when it is taken as the underlying logic, Peano arithmetic is categorical. Slightly augmented first-order logic is similar to the system described in [Scott *1979*], except that the latter is a constructive theory of types, has infinitely many free variables in each type, and imposes the generality interpretation on free variables in every type.

3. Full Explicit Definitions. We understand explicit, equational, and conditional definitions. as they are typically presented in formalizations of standard first-order logic. Recall that the conditions on explicit and equational definitions insure that defined symbols are *eliminable* in favor of the primitive symbols and that definitions are *noncreative* in the sense that, after the definition is introduced, no previously unprovable assertion containing only logical and primitive symbols becomes provable. We summarize the presentation of Suppes ([Suppes *1957*]), modifying his treatment of free variables to ease the transition to the free case.[19]

Let $n \geq 0$, f^n be an n-ary function symbol (function parameter), $t(a_1, \ldots, a_n)$ be a term, and

$$f^n(a_1, \ldots, a_n) = t(a_1, \ldots, a_n)$$

be an equation. We call its universal closure

$$\forall X_1 \ldots \forall X_n (f^n(X_1, \ldots, X_n) = t(X_1, \ldots, X_n))$$

introducing the new n-ary function symbol f^n an *equational definition* and f^n a *defined symbol* if the following conditions are satisfied:

1. the X_i $(1 \leq i \leq n)$ are distinct individual variables,

2. the X_i are the only individual variables occurring in $t(X_1, \ldots, X_n)$,

and

3. the only nonlogical symbols occurring in $t(X_1, \ldots, X_n)$ are either primitive symbols or previously defined symbols.

For example, in arithmetic, the individual constant 1 and the 1-ary function plus2 can be introduced with the equational definitions:

$$1 = \text{succ}(0)$$

and

$$\forall X \ (\text{plus2}(X) = \text{succ}(\text{succ}(X)))$$

An *explicit definition* of an n-ary predicate p^n is of the form:

---

[19] Suppes takes definitions to be open sentences in which free variables are given the generality interpretation. Simply put, our definitions are the universal closure of his definitions.

$$\forall X_1 \ldots \forall X_n \, (p^n (X_1, \ldots , X_n) \leftrightarrow A(X_1, \ldots, X_n))$$

if the following conditions are satisfied:

1. the X_i $(1 \leq i \leq n)$ are distinct individual variables,
2. the X_i are the only individual variables occurring in $A(X_1, \ldots , X_n)$, and
3. the only nonlogical symbols occurring in $A(X_1, \ldots, X_n)$ are either primitive symbols or previously defined symbols.

For example, in arithmetic $\leq$ can be defined as follows:

$$\forall X \, \forall Y \, (X \leq Y \leftrightarrow \exists Z \, (X + Z = Y))$$

Similarly, an *explicit definition* of an n-ary function f^n is of the form:

$$\forall X_1 \ldots \forall X_n \, \forall Y(f^n(X_1 \ldots \forall X_n) = Y \leftrightarrow A(X_1 \ldots \forall X_n, \mathrm{Y})$$

if the following conditions are satisfied:

1. the X_i $(1 \leq i \leq n)$ and Y are distinct individual variables,
2. the X_i and Y are the only individual variables occurring in $A(X_1 \ldots \forall X_n, Y)$,
3. the only nonlogical symbols occurring in $A(X_1 \ldots \forall X_n, Y)$ are either primitive symbols or previously defined symbols, and
4. the existence and uniqueness conditions:

$$\forall X_1 \ldots \forall X_n \, \exists Y \, A(X_1 \ldots \forall X_n, Y)$$
$$\forall X_1 \ldots \forall X_n \, \forall Y' \, A(X_1 \ldots \forall X_n, Y) \wedge A(X_1 \ldots \forall X_n, Y') \rightarrow Y = Y')$$

are provable from the nonlogical axioms and preceding definitions.

An equational definition is equivalent to an explicit definition. For example, 1 and plus2 can be defined in arithmetic with the explicit definitions:

$$\forall Y(1 = Y \leftrightarrow Y = \mathrm{succ}\ (0))$$

and

$$\forall X \, \forall Y \, (\mathrm{plus2}(X) = Y \leftrightarrow Y = \mathrm{succ}(\mathrm{succ}(X)))$$

The requirement that existence and uniqueness conditions be provable is omitted from equational definitions because these conditions are provable without exception for equational definitions.

The treatment of conditional definitions in standard logic is messy and provides some of the motivation for free logic definition theory. A *conditional definition* of an n-ary function f^n of the form:

$$\forall X_1 \ldots \forall X_n \, \forall Y \, (C(X_1 \ldots X_n) \rightarrow (f^n (X_1 \ldots X_n = Y \leftrightarrow A(X_1 \ldots \forall X_n, Y)$$

if the following conditions are satisfied:

1. the X_i $(1 \leq I \leq n)$ and Y are distinct individual variables,
2. the X_i and Y are the only individual variables occurring in $A(X_1, \ldots, X_n, Y)$,
3. the only nonlogical symbols occurring in $A(X_1, \ldots, X_n, Y)$ are either primitive symbols or previously defined symbols, and
4. the conditional existence and uniqueness conditions:

$$\forall X_1 \ldots, \forall X_n \, (C(X_1, \ldots, X_n) \rightarrow \exists Y \, A(X_1 \ldots \forall X_n, Y))$$
$$\forall X_1 \ldots, \forall X_n \, \forall Y \, \forall Y' \, (C(X_1, \ldots, X_n) \rightarrow A(X_1, \ldots, X_n, Y) \wedge A(X_1 \ldots \forall X_n, Y') \rightarrow Y = Y'))$$

are provable from the nonlogical axioms and preceding definitions.

If an individual constant, say 0, is primitive or has been previously defined, a conditional definition of the above form can be converted into the explicit definition:

$$\forall X_1 \ldots \forall X_n \, \forall Y \, (f^n (X_1 \ldots X_n = Y \leftrightarrow (C(X_1, \ldots, X_n) \rightarrow A(X_1 \ldots \forall X_n, Y)) \wedge \neg C (X_1 \ldots \forall X_n, Y') \rightarrow Y = 0))$$

For example, the conditional definition of division $(\div)$ in the theory of fields:

$$\forall X \, \forall Y \, \forall Z \, (Y \neq 0 \rightarrow (X \div Y = Z \leftrightarrow X = Y \times Z))$$

can be converted into the explicit definition:

$$\forall X \, \forall Y \, \forall Z \, (X \div Y = Z \leftrightarrow (Y \neq 0 \rightarrow X = Y \times Z) \wedge (Y = 0 \rightarrow Z = 0))$$

The definition theory for nonstrict positive free logic builds directly upon that for standard logic.[20] Explicit definitions in nonstrict free logic are of limited interest, because, like conditional definitions in standard logic, they are not sufficient for insuring eliminability in the general case. The theory of definitions in standard logic requires some modification to suit the free case. We require a device that, intuitively, enables us to quantify universally over the outer domain. The *universal expansion* $\forall^* a_1 \ldots \forall^* a_n A(a_1, \ldots, a_n)$ of a sentence $A(a_1, \ldots, a_n)$ is defined by induction on $n \geq 0$:

$$A() \equiv A$$

and

$$\forall^* a_1 \ldots \forall^* a_{n+1} A(a_1, \ldots, a_{n+1}) \equiv$$
$$\forall^* a_1 \ldots \forall^* a_n \, (\forall X_{n+1} A(a_1, \ldots, a_n, X_{n+1}) \wedge A(a_1, \ldots, a_n, \text{err})$$

We understand a *full explicit definition* of the predicate p^n to be the universal expansion $\forall^* a_1 \ldots \forall^* a_n A(a_1, \ldots, a_n)$ of a sentence A of the form

$$p^n(a_1, \ldots, a_n) \leftrightarrow B(a_1, \ldots, a_n)$$

Note that the universal closure of A is an explicit definition. For function symbols, the definitions of a *full explicit definition* and a *full equational definition* are entirely similar, except that the requirement in standard logic regarding the provability of the existence and uniqueness conditions must be modified. In nonstrict positive free logic, the analog of the existence condition is the universal expansion of:

$$\exists Y A(a_1, \ldots, a_n, Y) \wedge A(a_1, \ldots, a_n, \text{err})$$

and the analog of the uniqueness condition is the universal expansion of:

$$A(a_1, \ldots, a_n, b) \wedge A(a_1, \ldots, a_n, b') \rightarrow b = b'$$

---

[20] There has been some previous work in the theory of definitions in free logic. For example, [Schock *1965*] formulated rules of definition for strict predicates and functions as does [Renardel *1994*] in an infinitary free logic, and [Dwyer *1988*] studies the eliminability problem restricted to closed sentences. To enable the definition of nonstrict functions such as 'zero' — to be given shortly — nonstrict positive free logic requires a more general approach. We emphasize that the theory of definitions in nonstrict positive free logic, especially that presented in Section 5, is facilitated by the "inner domain, outer domain" semantics.

In a *full explicit definition* of a function symbol, we require that the analogs of the existence and uniqueness conditions be proven prior to the introduction of the symbol.[21] A straightforward argument, using the fact that explicit definitions are noncreative in standard logic, establishes that, in nonstrict positive free logic, full explicit definitions are noncreative and that predicate symbols introduced in full explicit definitions are eliminable.[22] In the case of a full explicit definition of a function symbol, proof that the function symbol is eliminable uses the fact that the the analog of the existence condition is provable.[23] In summary, we have:

Theorem 1. *A full explicit definition of a predicate (function) symbol is noncreative.*

Theorem 2. *A predicate (function) symbol introduced in a full explicit definition is eliminable.*

---

[21] Note that, although the axiom for err

$$\text{err} = t \leftrightarrow \neg\exists X\,(X = t)$$

appears to be equivalent to a full explicit definition, it is not because the analog of the existence condition for err cannot even be expressed, much less proven. This is because the system obtained by dropping this axiom and adding the axiom of "universal existence" $\exists X\,(X = t)$ is consistent. The models of this system are models of standard logic, because the inner and outer domains coincide. The undefinability of err is a curiosity arising from not having quantification over the outer domain.

[22] This is because a nonstrict free theory can be mapped, using the translation that can be read off from the definition of truth, into a standard theory in the metalanguage.

[23] The case for a 1-ary function symbol f illustrates the key step in the proof of eliminability. If f has been introduced with the full explicit definition (constituted by the universal expansion of the sentence):

$$f(a) = b \leftrightarrow B(a,b)$$

then f can be eliminated from $A(f(c))$ as follows:

$$\begin{aligned}
&A(f(c)) \Rightarrow \\
&\exists W\,(W = f(c) \wedge A(W)) \vee (\text{err} = f(c) \wedge A(\text{err})) \Rightarrow \\
&\exists W\,(B(c,W) \wedge A(W)) \vee (B(c,\text{err}) \wedge A(\text{err}))
\end{aligned}$$

The rest of the proof is as in the standard case. The case when f is n-ary is similar.

Since full explicit definition are noncreative and eliminable, we call them *completely proper.*

Somewhat analogous to a conditional definition, we also permit a *near-full explicit definition* of a function symbol, which is like a full explicit definition, except that we do not require that the analog of the existence condition be proven. We say that a near-full explicit definition is *proper* because it is noncreative. However, a function symbol introduced in a near-full explicit definition may not be eliminable.

The analog of the uniqueness condition for a full equational definition is provable making use of the laws of equality. The analog of the existence condition is also provable provided that the analog of the existence condition has been proven for every function symbol occurring in the definiens.

A predicate (function) introduced with a full explicit definition is called a *full* predicate (function), and one that is introduced with an explicit definition (but not a full explicit definition) *nonfull.* For example, $\leq$ is a nonstrict predicate because neither the sentence err $\leq$ err nor its negation is intuitively valid. Hence, it is a nonfull predicate because it is introduced in free arithmetic with the explicit definition given above. Furthermore, $\leq$ is a nonlogical constant because $<$ occurs in the definiens. However, $\neq$, E!, $\approx$, and $\not\approx$ are full predicates that are nonprimitive logical constants because they are introduced with the universal expansions of the following sentences:

$$a \neq b \leftrightarrow \neg(a = b)$$

$$\mathrm{E!}(a) \leftrightarrow \exists X\,(X = a)$$

$$a \approx b \leftrightarrow \mathrm{E!}(a) \wedge a = b$$

$$a \not\approx b \leftrightarrow \mathrm{E!}\,(A) \wedge \mathrm{E!}(B) \wedge a \neq b$$

Note that $\neq$ is a nonstrict predicate, while E!, $\approx$, and $\not\approx$ are strict.

The nonstrict function 'zero' is introduced with a full equational definition (constituted by the universal expansion of the sentence):

$$\mathrm{zero}\,(a) = 0$$

and 'if-then-else' with the full explicit definition

if a then b else $b' = c \leftrightarrow$

$$(a = 1 \rightarrow c = b) \wedge (a = 0 \rightarrow b' = c) \wedge (\neg(a = 0 \vee a = 1) \rightarrow c = \mathrm{err})$$

The analogs of the existence and uniqueness condition are readily proven for 'if-then-else', and so its definition is completely proper. The function symbols 'zero' and 'if-then-else' are eliminable nonlogical constants.

4. The Beth Definability Theorem. In standard logic, the Craig Interpolation Lemma implies the Beth Definability Theorem, and this result carries over to nonstrict positive free logic. In addition to its usual role in standard logic of insuring a reasonable model theory, the Craig Lemma in computer science has an additional, fundamental application. Software engineers make critical use of the Craig Lemma in establishing the Modularization Theorem which enables computer systems to be decomposed into modules [Parnas *1972*].[24]

Proof of the Craig Lemma can be had by modifying the proof for $\mathbf{CQ}^*_=$ in [Gumb *1979b*]. To accommodate the unique error object **err**, we supplement the usual clauses in the definition of a (Hintikka) model set with clauses stating, intuitively, that **err** cannot be an existent and that it is the unique error object. Let t and t' be any terms, A any sentence, and $\prec$ be a well-ordering of the singular terms with err as the first element. We understand a *model set* to be a set of sentences S satisfying the following conditions:[25]

---

[24] Turski and Maibaum ([Turski and Maibaum *1987*, 175] write: "It seems that the crucial requirement to establish the (Modularization Theorem) is that the Craig Interpolation Lemma holds in the linguistic system." See also [Veloso *1993*].

[25] In our definition of a model set, we follow the conventions in [Gumb *1979a*, 37–38], and [Gumb *1984*, 176]. Rules of the tree method are read off from the clauses defining a model set as usual. Corresponding adjustments in the rules of the tree method and its Routine in [Gumb 1979b, 322–323], are needed to suit the modified rule (D.=) and the new rule (I.$\overline{\varnothing}$), as well as the the new rules (C.err) and (D.err). The adjustment to rule (D.=) is required to handle function parameters. Roughly, we substitute one term for another term that is equal to it just when the first term is "simpler" ($\prec$) than the second. This blocks certain infinite loops in the Routine when we have, for example, $a = f(a)$.

The rule (I.$\overline{\varnothing}$) is an "initialization" rule and is fundamentally different in character from the C-rules and the D-rules. Unlike the C-rules, it does not directly signal that S cannot be a model set. Unlike the D-rules, it does not reflect the compositionality principles inherent in the semantics. The Routine accommodates it by requiring that, in Step 1 on p. 323, $\exists X\,(X = X)$ be placed at the top of the tree (and the line numbering changed accordingly). It insures that the inner domain of a model constructed from an open branch is nonempty as mandated by the semantics, and it blocks, for example, $\forall X\,(X \neq X)$ from being a member of a model set. Recall that in our semantics, we have $\models \exists X\,(X = X)$.

(I.$\overline{\varnothing}$) $\exists X\,(X = X) \in S$.
(C.$\neq$) $t \neq t \notin S$.
(C.$\neg$) If $A \in S$, then $\neg A \notin S$.
(C.err) $\exists X\,(X = \text{err}) \notin S$.
(D.=) If $A(t)$ is a literal or an existential, $A(t) \in S$, $t' \prec t$ and either $t = t' \in S$ or $t' = t \in S$, then $A(t') \in S$.
(D.err) If $\forall X\,(X \neq t) \in S$ or $\forall X\,(t \neq X) \in S$, then $t = \text{err} \in S$.
(D.$\neg\neg$) If $\neg\neg A \in S$, then $A \in S$.
(D. $\neg\rightarrow$) If $\neg\,(A \rightarrow B) \in S$, then $A, \neg B \in S$.
(D.$\rightarrow$) If $A \rightarrow B \in S$, then either $\neg A \in S$ or $B \in S$.
(D.$\neg\forall$) If $\neg\forall X\,A(X) \in S$ and $\neg\forall X\,A(X) \neq \exists X(X = t')$ for some term t', then, for some term t, $A(t)$, $\exists X(X = t) \in S$.
(D.$\forall$) If $\forall X\,A(X) \in S$, then $A(t) \in S$ for each term t such that $\exists X(X = t) \in S$.

The following result is the key ingredient in proof of the completeness of the tree method for nonstrict positive free logic:[26]

Theorem 3. (Hintikka's Lemma): *A model set is satisfiable.*

Proof of the following is had by a straightforward editing of the proof in [Gumb *1979b*]:[27]

Theorem 4. Extended Joint Consistency Theorem (*Craig-Lyndon-Robinson*): The theory $T_1 \cup T_2$ *is inconsistent iff there is a sentence F such that*

Illustrating the other new rules, rule (C.err) blocks the unsatisfiable $\exists X\,(X = \text{err})$ from being a member of a model set. We require that err be the first element in the well-ordering $\prec$ so that there is no $t \prec \text{err}$, blocking $\{\exists X\,(X = t),\ t = \text{err}\}$ from being included in a model set. As it is, we have $\{\exists X\,(X = t),\ t = \text{err},\ \exists X\,(X \neq \text{err})\}$ by rule (D.=), and now rule (C.err) can be applied. The rule (D.err) in conjunction with (C.err) blocks $\{\forall X\,(X \neq t), \forall X\,(X \neq t'),\ t \neq t'\}$. The motivation for the other rules is as usual.

[26] This proof is easily had by editing the proofs in [Gumb*1979a*], 38–43, with modifications to suit our model-theoretic semantics as in Boolos and Jeffrey's simple completeness proof [Boolos & Jeffrey *1974*, 138–143

[27] The case for the C-rule (C.err) is incorporated into the inductive proof of Case 1, as part of the Basis, Case 1, on p. 324. The cases for the introduction rule (I.$\overline{\varnothing}$) and the D-rule (D.err) are treated in the inductive step like Case 1.1 for (D. $\neg\neg$) on p. 330. A model-theoretic proof of a result similar to that in [Gumb *1979b*] can by found [Dwyer *1988*].

1. $T_1 \cup \{F\}$ and $T_2 \cup \{\neg F\}$ *are inconsistent,*

2. *each sentence and predicate parameter occurring in F also occurs in both* T_1 *and* T_2, *and*

3. *each sentence and predicate parameter occurring in F occurs in* T_1 *the opposite way as in F and in* T_2 *the same way as in F.*

We now present a version of Beth's theorem that incorporates justification for Padoa's method. Let p^n, $p_1^{n_1}, \ldots, p_m^{n_m}$ be distinct predicates of arity $n, n_1, \ldots, n_m$ occurring in the theory T, and

$$(\mathrm{D}.\, p^n) \equiv \forall^* a_1 \ldots \forall^* a_n \, (p^n (a_1 \ldots a_n) \leftrightarrow (B\, (a_1 \ldots a_n))$$

The predicate p is called *fully and explicitly definable* from $p_1, \ldots, p_m$ in T if $\vdash_T$ (D.p^n). Clearly, if p were new to the theory T^- and $T = T^- \cup$ (D.p^n), then p would be fully and explicitly definable in T. An entirely similar terminology is used in the case of functions. Let q^n be a new predicate, and T_q be the result of substituting q^n for every occurrence of p^n in T. The predicate q^n is *implicitly definable* from $p_1, \ldots, p_m$ in T if

$$T \cup T_q \vdash \forall^* a_1 \ldots \forall^* a_n \, (p^n \, (a_1, \ldots, a_n) \leftrightarrow (q^n \, (a_1, \ldots, a_n))$$

Proof of the Beth Definability Theorem as found in the literature (see, for example [Boolos and& Jeffrey *1974*] and [Smullyan *1968*]) is readily adapted to the nonstrict positive free case. The following proof is typical of those used for obtaining the results reported in this paper in that the modifications to the proofs for standard logic are routine. However, in the literature, the Craig Interpolation Lemma is normally used, and, because the Extended Joint Consistency Theorem (Theorem 4) can be used directly instead, we briefly outline the proof to show how the Theorem slightly simplifies the proof by obviating appeal to the Compactness Theorem.

Theorem 5. Beth Definability Theorem: *A predicate (function) is implicitly definable iff it is explicitly definable.*

Proof: To reduce clutter, we follow Smullyan [Smullyan *1968*, 132–133) in restricting attention to 1-ary predicates. We prove

$$\text{(ED) } T \vdash \forall^* a \, (p\, (a) \leftrightarrow (B\, (a))$$

$$\Leftrightarrow$$

$$\text{(ID) } T \cup T_q \vdash \forall^* a_1 \ldots \forall^* a \, (p\, (a) \leftrightarrow (q\, (a))$$

($\Rightarrow$) The justification of Padoa's method is very simple. Suppose (ED). Then $T_q \vdash \forall^* a\ (q\ (a) \leftrightarrow (B\ (a))$. Hence, (ID) because p and q are foreign to T and $B(a)$.

($\Leftarrow$) Suppose (ID). Letting a be an individual parameter foreign to T, we have:

$$T \cup T_q \cup \{\exists X\ (X = a\ )\} \vdash (p\ (a) \leftrightarrow (q\ (a)) \wedge (p(\text{err}) \leftrightarrow q(\text{err}))$$

Hence, the two theories

1. $\{\exists X\ (X = a\ )\} \cup T \cup \{\neg\, p(a)\}) \cup \{\exists X\ (X = a\ )\} \cup T_q \cup \{q\ (a)\})$

and

2. $T \cup \{\neg\, p(\text{err})\}) \cup T_q \cup \{q(\text{err})\})$

are inconsistent. By Theorem 4, there are F and F_{err} as in the theorem such that each of the following theories is inconsistent:

3. $\{\exists X\ (X = a\ )\} \cup T \cup \{\neg\, p(a)\} \cup \{F(a)\}$

4. $\{\exists X\ (X = a\ )\} \cup T_q \cup \{q(a)\} \cup \{\neg F(a)\}$

5. $T \cup \{\neg\, p(\text{err})\} \cup \{F_{\text{err}}(\text{err})\}$

6. $T_q \cup \{\ q(\text{err})\} \cup \{\ \neg\, F_{\text{err}}(\text{err})\}$

From (4) and (6), we have that (7) and (8) are inconsistent:

7. $\{\exists X\ (X = a\ )\} \cup T \cup \{p(a)\} \cup \{\neg F(a)\}$

8. $T \cup \{p(\text{err})\} \cup \{\neg\, F_{\text{err}}(\text{err})\}$

Assembling (3), (5), (7), and (8), we have (9) and hence (10):

9. $T \vdash \forall X\ (p(X) \leftrightarrow F(X)) \wedge (p(\text{err}) \leftrightarrow F(\text{err}))$

10. $T \vdash \forall^* a\ (p(a) \leftrightarrow (a \neq \text{err} \rightarrow F(a)) \wedge (a = \text{err}) \rightarrow F_{\text{err}}(\text{err})))$

Taking $B(a) \equiv (a \neq \text{err} \rightarrow F(a)) \wedge (a = \text{err}) \rightarrow F_{\text{err}}(\text{err})))$ in (10), we see that we have proven (ED). □

5. Definitional Transformations. In this section, we describe transformations on sentences that yield full explicit definitions. Under some of these transformations, results in standard logic carry over into nonstrict positive free logic. This is useful because some previously established theorems about definitional extensions of standard theories can be had in the nonstrict positive free case without proving each individual result "from scratch." Using these transformations, many facts about mathematical theories that are adequately formalized in standard logic can be easily had in nonstrict positive free logic, a logic that is also adequate for formalizing computing practice.

The first two transformations convert definitions in nonstrict positive free logic that are "not quite" full explicit definitions into ones that are. The *equational transformation* of a full equational definition into a full explicit definition is schematized as follows:

$$\forall^* a_1 \ldots \forall^* a_n\, (f^n(a_1, \ldots, a_n) = t((a_1, \ldots, a_n)) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n\, \forall^* b\, (f^n(a_1, \ldots, a_n) = b \leftrightarrow b = t((a_1, \ldots, a_n))$$

For example, we have for the nonstrict function 'zero'

$$\forall^* a\, (\text{zero}(a) = 0) \equiv \forall X\, (\text{zero}(X) = 0 \wedge \text{zero}(\text{err}) = 0 \Rightarrow$$
$$\forall^* a\, \forall^* b\, (\text{zero}(a) = b \leftrightarrow b = 0) \equiv$$
$$\forall X\, \forall Y\, (\text{zero}(X) = Y \leftrightarrow X = 0 \wedge \forall X\, (\text{zero}(X) = \text{err} \leftrightarrow \text{err} = 0) \wedge$$
$$\forall Y\, (\text{zero}(\text{err}) = Y \leftrightarrow Y = 0) \wedge (\text{zero}(\text{err}) = \text{err} \leftrightarrow \text{err} = 0)$$

A full explicit definition obtained by applying the equational transformation is equivalent to the original full equational definition.

Consider the strict 1-ary functions id (the identity function), twice (the doubling function),[28] and $\perp$ (the totally undefined function). The reader might find it instructive to compare the role of the four conjuncts in the full explicit definition of 'zero' with the roles of the corresponding four conjuncts in the full explicit definitions obtained by equational transformation from the following full equational definitions:

$$\text{id}(a) = a$$

[28] The function twice is strict because + is understood to be strict. See, for example, the axiomatization of arithmetic in [Gumb *1989*, Chapter 5], as corrected in [Gumb *1994*] and [Gumb *1996*].

$$\text{twice}(a) = a + a$$

$$\perp(a) = \text{err}$$

In standard logic, the functions zero and id can be introduced with entirely similar equational definitions, but $\perp$ cannot be introduced at all because it is not a total function.

The *near-full explicit transformation* of a near-full explicit definition of an n-ary function f^n into a full explicit definition is schematized as follows:

$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\, (f^n(a_1, \ldots, a_n) = b \leftrightarrow A((a_1, \ldots, a_n))) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\, (f^n(a_1, \ldots, a_n) = b \leftrightarrow$$
$$\exists Y\, A((a_1, \ldots, a_n, Y) \rightarrow b = Y) \wedge (\neg\, \exists\, Y A(a_1, \ldots, a_n, Y) \rightarrow b = \text{err}$$

The analog of the uniqueness condition holds for the full explicit definition because it holds for the near-full explicit definition, and proof of the analog of the existence condition follows immediately from the form of the full explicit definition.

We say that a (nonstrict positive) free theory *corresponds* with a standard theory provided that the free theory shares the same nonlogical vocabulary as the standard theory and the err-free fragment of the set of theorems of the free theory is identical with the set of theorems of the standard theory. The next transformation converts a standard theory into a free theory that corresponds with the standard theory.

The *strict axiom transformation* of a standard theory T is obtained by supplementing T with axioms stating that each primitive n-ary function ($n \geq 0$) is total:

$$\forall X_1 \ldots \forall X_n\, \exists\, Y f^n(X_1, \ldots, X_n,) = Y$$

and strict:

$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\, (\neg(\text{E!}\, (a_1 \wedge \ldots \wedge (\text{E!}\, (a_n,)) \rightarrow f^n(a_1, \ldots, a_n,) = \text{err}$$

and each primitive n-ary predicate p^n is strict:

$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\, (\neg(\text{E!}\, (a_1 \wedge \ldots \wedge (\text{E!}\, (a_n,)) \rightarrow p^n(a_1, \ldots, a_n,))$$

The next set of transformations preserve correspondence in definitional extensions.

1. The *strict transformation* of an explicit definition in standard logic of an n-ary predicate p^n into a full explicit definition is schematized as follows:

$$\forall X_1 \ldots \forall X_n\,(p^n\,(X_1, \ldots, X_n) \leftrightarrow A((X_1, \ldots, X_n)) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n\,(p^n\,(a_1, \ldots, a_n) \leftrightarrow$$
$$\mathrm{E!}\,(a_1) \wedge \ldots \wedge \mathrm{E!}\,(a_n) \wedge A(a_1, \ldots, a_n))$$

2. The *strict transformation* of an explicit definition in standard logic of an n-ary function f^n into a full explicit definition is schematized as follows:

$$\forall X_1 \ldots \forall X_n\,\forall Y f^n\,(X_1, \ldots, X_n) = Y \leftrightarrow A((X_1, \ldots, X_n)) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n \forall b\,(f^n\,(a_1, \ldots, a_n) = b \leftrightarrow$$
$$(\mathrm{E!}\,(a_1) \wedge \ldots \wedge \mathrm{E!}\,(a_n) \wedge A(a_1, \ldots, a_n, b) \wedge$$
$$(\neg(\mathrm{E!}\,(a_1 \wedge \ldots \wedge (\mathrm{E!}\,(a_n)) \rightarrow b = \mathrm{err}))$$

3. The *strict transformation* of a conditional definition in standard logic of an n-ary function f^n into a full explicit definition is schematized as follows:[29]

$$\forall X_1 \ldots \forall X_n \forall Y\,(C\,(X_1, \ldots, X_n) \rightarrow$$
$$(f^n\,(X_1, \ldots, X_n) = Y \leftrightarrow A(X_1, \ldots, X_n, Y))) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\,(f^n\,(a_1, \ldots, a_n) = b \leftrightarrow$$
$$(\mathrm{E!}(a_1) \wedge \ldots \wedge \mathrm{E!}(a_n) \wedge C(a_1, \ldots, a_n) \rightarrow (a_1, \ldots, a_n, b)) \wedge$$
$$(\neg\,(\mathrm{E!}(a_1) \wedge \ldots \wedge \mathrm{E!}(a_n) \wedge C(a_1, \ldots, a_n)) \rightarrow b = \mathrm{err})$$

For example, the conditional definition of division ($\div$) in the theory of fields can be converted to a full explicit definition as follows:[30]

---

[29] The strict transformation is related to a device of Schock [Schock *1965*, 40]. Schock points out that with a transformation of the following form:

$$\forall X_1 \ldots \forall X_n \forall Y(C(X_1, \ldots, X_n) \rightarrow (f^n(X_1, \ldots, X_n) = Y \leftrightarrow A(X_1, \ldots, X_n, Y))) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n \forall^* b(f^n(a_1, \ldots, a_n) = b \leftrightarrow$$
$$C(a_1, \ldots, a_n) \wedge A(a_1, \ldots, a_n, b))$$

it follows that

$$\vdash_T \forall X_1 \ldots \forall X_n\,(\neg\,C(X_1, \ldots, X_n) \rightarrow \neg\,\mathrm{E!}(f^n\,(X_1, \ldots, X_n)))$$

$$\forall X \forall Y \forall Z\,(Y \neq 0 \rightarrow (X \div Y = Z \leftrightarrow X = Y \times Z)) \Rightarrow$$
$$\forall^* a \ldots \forall^* b \forall^* c\,(a \div b = c \leftrightarrow$$
$$(\mathrm{E!}(a) \wedge \mathrm{E!}(b) \wedge \mathrm{E!}(c)\ b \neq 0 \rightarrow a = b \times c) \wedge$$
$$(\neg\,(\mathrm{E!}(a) \wedge \mathrm{E!}(b) \wedge \mathrm{E!}(c) \wedge b \neq o) \rightarrow b = \mathrm{err}))$$

The applicability of this transformation is not subject to a restriction similar to the restriction on the corresponding transformation in standard logic discussed in Section 3, because the error constant is always present.

4. The *closure transformation* of an explicit definition in standard logic of an n-ary predicate p^n into a full explicit definition is schematized as follows:

$$\forall X_1 \ldots \forall X_n\,(p^n\,(X_1, \ldots, X_n) \leftrightarrow A(X_1, \ldots, X_n)) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n\,(p^n\,(a_1, \ldots, a_n) \leftrightarrow A(X_1, \ldots, X_n))$$

5. The *closure transformation* of an equational definition in standard logic of an n-ary function f^n into a full equational definition is schematized as follows:

$$\forall X_1 \ldots \forall X_n\,(f^n\,(X_1, \ldots, X_n) = t\,(X_1, \ldots, X_n)) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\,(f^n\,(a_1, \ldots, a_n) = b \leftrightarrow b = t\,(a_1, \ldots, a_n))$$

Proof of the following result can be extracted from the literature:[31]

Theorem 6. *(a) If a free theory is the strict axiom transformation of a standard theory then it corresponds to the standard theory. (b) If T is a free theory that corresponds with the standard theory T_S, T′ is obtained by supplementing T with a full explicit definition D, T'_S is obtained by T_S with*

---

[30] Our example follows Dwyer [Dwyer *1988*, 51], who uses Schock's device to supplement the theory of fields, yielding the theory F_+ of fields with explicit division. Dwyer notes that $\vdash F_+ \forall X \neg \mathrm{E!}(X \div 0)$.

[31] These proofs are obtained by slightly modifying standard results of first-order logic as presented in [Suppes *1957*] and [Schoenfield *1967*]. Regarding the strictness transformations, see Farmer's Eliminability Theorem for his partial first-order logic (Farmer *1995*], and his remark in [Farmer *1990*, 1289]. Regarding the strictness transformation of conditional sentences, also see [Schock *1965*]. Regarding the closure transformation of an equational definition, an equational definition can be transformed into a full equational definition as an intermediate step, and the result then follows from the remarks made earlier in this paper.

an explicit definition D_S, and D is obtained from D_S by any of the following transformations:

> *1. the strict transformation of an explicit definition in standard logic of a predicate symbol,*
> *2. the strict transformation of an explicit definition in standard logic of a function symbol,*
> *3. the strict transformation of a conditional definition in standard logic of a function symbol,*
> *4. the closure transformation of an explicit definition in standard logic of a predicate symbol,*
> *5. the closure transformation of an equational definition in standard logic of a function symbol, and*

then T' corresponds with , T'_S.

The final transformation applies to an explicit definition of function symbol, and it is more problematic in that we require a precondition insuring that the analog of the uniqueness condition holds in the resulting full explicit definition. The precondition *blocks* application of the transformation when the the analog of the uniqueness condition is not guaranteed. We first exhibit the transformation, and then present the precondition. The *closure transformation* of an explicit definition in standard logic of an n-ary function f^n into a full explicit definition is schematized as follows:

$$\forall X_1 \ldots \forall X_n \forall Y\, (f^n(X_1, \ldots, X_n) = Y \leftrightarrow A(X_1, \ldots, X_n)) \Rightarrow$$
$$\forall^* a_1 \ldots \forall^* a_n \forall^* b\, (f^n(a_1, \ldots, a_n) = b \leftrightarrow A(a_1, \ldots, a_n, b))$$

In general, this closure transformation does not insure that the analog of the uniqueness condition is provable for the resulting full explicit definition. We seek a natural precondition guaranteeing that the resulting full explicit definition is completely proper. In other words, the precondition is to insure that the analogs of the existence and uniqueness conditions hold for the full explicit definition. It is readily verified that the analog of the existence condition holds. Consider the closure transformation of a 1-ary function f:

$$\forall X \forall Y\, (f(X) = Y \leftrightarrow A(X, Y)) \Rightarrow$$
$$\forall^* a \forall^* b\, (f(a) = b \leftrightarrow A(a, b)) \equiv$$
$$\forall X \forall Y\, (f(X) = Y \leftrightarrow A(X, Y)) \wedge \forall X\, (f(X) = \text{err} \leftrightarrow A(X, \text{err})) \wedge$$
$$\forall Y\, (f(\text{err}) = Y \leftrightarrow A(\text{err}, Y)) \wedge (f(\text{err}) = \text{err} \leftrightarrow A(\text{err}, \text{err}))$$

Since f is a total function in standard logic, we have $\vdash_T \forall X \exists Y A(X, Y)$ and $\forall X \forall Y \forall Y'\, (A(X, Y) \wedge A(X, Y') \rightarrow Y = Y')$. However, the second, third, and fourth conjuncts do not block, for example, $\vdash_T \exists X\, A(X, \text{err})$, which would give the absurdity $\vdash_T \exists X\, (X = \text{err})$, However, if there are at least two

existents $\vdash_T \exists X \exists Y\,(X \neq Y)$), the following *monotonicity condition* insures the uniqueness condition:

$$\vdash_T \forall X\,(A(X, \text{err}) \rightarrow \exists Y\,A(X, Y) \rightarrow \forall Y\,A(X, Y))) \wedge$$
$$\forall X\,A(\text{err}, \text{err}) \rightarrow (\exists Y\,A(\text{err}, Y) \rightarrow \forall Y\,A(X, Y)))$$

Some examples may clarify how the requirements of at least two existents and the monotonicity condition block inappropriate applications of the closure transformation. Taking the existence of at least two individuals as a precondition blocks the closure transformation from being applicable to the explicit definition:

$$\forall X\,\forall Y\,(f(X) = Y \leftrightarrow\ Y = Y)$$

which is a proper explicit definition in any standard theory T in which there is exactly one individual ($\vdash_T\ \exists X\,\forall Y\,(X = Y)$). The monotonicity condition blocks the closure transformation from being applicable to the explicit definition:

$$\forall X\,\forall Y\,(f(X) = Y \leftrightarrow\ X \neq Y)$$

which is a proper explicit definition in any standard theory T in which there are exactly two individuals ($\vdash_T \exists X\,\exists Y\,(X \neq Y \wedge \forall Z\,(Z = X \vee Z = Y))$). On the other hand, the closure transformation is applicable to the explicit definition of the function ζ

$$\forall X\,\forall Y\,(\zeta(X) = Y \leftrightarrow\ y = 0)$$

in any theory in which there are at least two individuals ($\vdash_T\ \exists X\,\exists Y\,(X \neq$ Y)) because the monotonicity condition is satisfied, and yields the full explicit definition of the function zero given earlier in this section.

More generally for n-ary function f^n the *monotonicity condition* is given by the schema:

$$\vdash_T \forall X_1 \ldots \forall X_n\,(A\,(X_1, \ldots, X_n, \text{err})[\text{err}\backslash X_{i_1}] \ldots [\text{err}\backslash X_{i_m}] \rightarrow$$
$$(\exists Y\,A\,(X_1, \ldots, X_n, Y)[\text{err}\backslash X_{i_1}] \ldots [\text{err}\backslash X_{i_m}] \rightarrow \forall Y A\,(X_1, \ldots, X_n, Y)))$$
$$(1 \leq i_k \leq n \wedge 1 \leq k \leq m)$$

We conclude that the *precondition* for the closure transformation must require that there be at least two existents and that the monotonicity condition hold. This precondition is a natural condition insuring that a full ex-

plicit definition obtained by the closure transformation is (completely) proper, and it works because the uniqueness condition holds for the given explicit definition in standard logic. In summary, we have:

Theorem 7. *If T is a free theory that corresponds with the standard theory T_S, T'_S is obtained by supplementing T_S with an explicit definition D_S of a function symbol, T' is obtained by supplementing T with a full explicit definition obtained by applying the closure transformation to D_S, and the precondition for the closure transformation is provable in T, then T' corresponds with T'_S.*

6. Limitations of Nonstrict Positive Free Logic. In nonstrict positive free logic as presented here, there is exactly one error object. We believe that this approach is entirely appropriate for program specification and verification. One error object is enough because the nonexistents are of no mathematical interest.[32] This approach is realistic because an expression in a program may be in error and may not refer to a conventional mathematical object. It is natural because the quantifiers range over conventional mathematical objects, freeing the axiomatization from the clutter of explicit clauses intended to handle errors.[33] Our approach can be generalized just in case the number of error objects is some definite positive integer. One might have two error objects, one for nontermination and one for division by zero. More generally, a finite number of nonidentical error objects might be provided for nontermination and each kind of exception that could be raised during a computation.[34] Just as a single error constant err must be primitive as discussed in footnote 21 above, in a slightly modified logic, each distinct error object would require a primitive error constant.

An outer domain quantifier, as studied by Cocchiarella [Cocchiarella *1966*], would be required to accommodate an infinite number of error objects. McLean [McLean *1990*], in his review of [Gumb *1989*], proposes a

---

[32] Scott makes a somewhat related point in [Scott *1970*, 146–147].

[33] In the higher-order case, caution must be exercised in that identifying error objects can lead to inconsistency. [Kuper *1994*, 108–109] notes that it is a well-known fact about the untyped λ–calculus that identifying all diverging computations leads to theidentification of all terms whatsoever. He considers the diverging terms Ω, $\lambda x.x\mathbf{K}\Omega$, and $\lambda x.x\mathbf{S}\Omega$, and, after applying the second and third to $\mathbf{K}$, finds that $\mathbf{K} = \mathbf{S}$.

[34] Having a definite, finite number of error objects can be motivated by other considerations. For example, Cartright and Felleisen [Cartright and Felleisen *1992*] have demonstrated that exactly two error objects in addition to the diverging object $\perp$ are needed to obtain a "fully abstract" semantics for a sequential version of Scott's programming language **PCF**.

semantics with an infinite number of error objects that is reminiscent of Meyer and Lambert [Meyer and Lambert *1968*]. He advocates that there be an error object for each possible syntactically distinct error in a program. These error objects might be useful, one might think, in programming debugging. Yet, how useful would these error objects be in isolation, without additional information on the context (environmental and otherwise) in which errors arise? At the low level of abstraction needed in program debugging, an infinite number of possible error messages might be entirely appropriate. However, at the higher levels of abstraction required for program specification and verification, a definite, finite number of error objects is sufficient, and, necessary for uncluttered reasoning.[35]

An infinite number of error objects cannot be handled adequately in free logic unless one is willing to have quantification over the outer domain, a tactic deplored by free logicians. Our position is that the (nonconstructive) positive free logic known in the literature as **E**-*logic* [Scott *1979*], [Troelstra and van Dalen *1988*]) is inadequate for formalizing the logic of program specification and verification. As noted by Lambert ([Lambert *1979*]), a principle of substitution *salva veritate*, essential to the theory of definition, is not forthcoming in E-logic. Further, E-logic purports to capture universal quantification over the outer domain by taking free variables to have the generality interpretation. As a deductive system for E-logic, Scott ([Scott *1979*, 663]) suggests the one in Dummett [Dummett *1977*}, 127]). However, Dummett's system makes no special provision for sentences with free variables given the generality interpretation. The Deduction Theorem[36] and Robinson Joint Consistency Theorem[37] fail in E-logic. E-logic cannot be employed in the manner intended and can be salvaged only by admitting quantifiers over the outer domain as advocated by Cocchiarella.[38]

---

[35] I am grateful to John Peterson for pointing out the need to distinguish the error requirements of different levels of abstraction.

[36] The Deduction Theorem fails because in E-logic we have, for example, $X = 0 \vdash X = 1$ but $\nvdash X = 0 \rightarrow X = 1$. This corresponds to the fact that, in standard logic, we have $\forall X\,(X = 0) \vdash \forall X\,(X = 1)$ while $\nvdash \forall X\,(X = 0 \rightarrow X = 1)$.

[37] The Robinson Theorem fails because, for example, $\{A(X)\} \cup \{\neg A(Y)\}$ is unsatisfiable in E-logic,but the separating sentence is not expressible. To see this, consider in standard first-order logic that $\forall X \ \{A(X)\} \cup \{\forall Y \neg A(Y)\}$ is unsatisfiable and that $\exists Z \neg A(Z)$ serves as a separating sentence.

[38] Davis and Fecher ([Davis and Fecher *1991*]) argue that existential quantification can be captured in a system with free variables given the generality interpretation, essentially a version of Hilbert's ε-calculus. However, this is not the approach taken in E-logic.

Bibliography

Michael **BEESON**. 1985. *Foundations of constructive mathematics*, Berlin/Heidelberg/New York/Tokyo, Springer-Verlag.

Ermanno **BENCIVENGA**. 1986. *Free logics*, D. Gabbay and F. Guenthner (editors), *Handbook of philosophical logic*, Vol. 3 (Dordrecht, Reidel), 373–426.

George S. **BOOLOS** and Richard C. **JEFFREY**. 1974. *Computability and logic*, Cambridge, Cambridge University Press.

Robert **CARTRIGHT** and Matthias **FELLEISEN**. 1992. *Observable sequentiality and full abstraction*, in *Conference Record of the Nineteenth Annual ACM Symposium on the Principles of Programming Languages*, ACM, 328–342.

Alonzo **CHURCH**. 1965. Review of [Lambert *1963*], The Journal of Symbolic Logic **30**, 103–104.

—. 1973. Review of [Scott *1967*], The Journal of Symbolic Logic **38**, 166–169.

Nino **COCHIARELLA**. 1966. *A logic of possible and actual objects*, The Journal of Symbolic Logic **31**, 688.

John **CORCORAN**.1973, *Gaps between logical theory and mathematical practice*, M. Bunge (editor), *The methodological unity of science* (Dordrecht, Reidel), 23–50.

—. 1980. *Categoricity*, History and Philosophy of Logic **1**, 187–207.

Martin **DAVIS** and Ronald **FECHER**. 1991. *A free variable version of the first-order predicate calculus*, Logic and Computation **1**, 431–451.

Gerard R. **RENARDEL DE LAVALETTE**. 1994. *From implicit via inductive to explicit definitions*, C. A. Middelburg, D. J. Andrews, J. F. Groote (editors), *Semantics of specification languages, Workshops in computing* (Berlin/ Heidelberg/New York/Tokyo, Springer-Verlag), 304–314.

Michael **DUMMETT**. 1977. *Elements of intuitionism*, Oxford, Oxford University Press.

Robert C. **DWYER**. 1988. *Denoting and defining: A study in free logic*, Ph.D. thesis, University of California, Irvine.

William M. **FARMER**. 1988. *A partial functions version of Church's simple theory of types*, Technical Report M88–52, MITRE Corporation, Bedford, Massachusetts, November 1988. (An earlier version of [Farmer *1990*] that uses $\perp$.)

—. 1990. *A partial functions version of Church's simple theory of types*, The Journal of Symbolic Logic **55**, 1269–1291.

—. 1995. *Reasoning about partial functions with the aid of a computer*, Erkenntnis, forthcoming. Presented at the Partial Functions and Programming Conference, University of California at Irvine, February 17, 1995.

Solomon **FEFERMAN**. 1992. *Logics for termination and correctness of functional programs*, Y. N. Moschovakis (editor), *Logics from computer science* (Berlin/Heidelberg/New York/Tokyo, Springer-Verlag), 95–127.

L. M. G. FEIJS and H. B. M. JONKERS. 1992. *Formal specification and design*, Cambridge, Cambridge University Press.

M.P. FOURMAN, C. J. MULVEY, and D. S. SCOTT (editors). 1979. *Applications of sheaves: Proceedings of the research symposium on applications of sheaf theory to logic, algebra, and analysis, Durham*, Lecture Notes in Mathematics, Vol. 753, Berlin, Springer-Verlag.

Dmitry P. GORSKY. 1981. *Definition*, Moscow, Progress Publishers.

Raymond D. GUMB. 1979a. *Evolving theories*, New York, Haven.

—. 1979b. *An extended joint consistency theorem for free logic with equality*, Notre Dame Journal of Formal Logic **20**, 321–335. Abstract in the Journal of Symbolic Logic **42** (1977), 146.

—. 1982. *On the underlying logics of specification languages*, ACM Software Engineering Notes **4**, 21–23.

—. 1984. *An extended joint consistency theorem for a family of free modal logics with equality*, The Journal of Symbolic Logic **49**, 174–183. Abstract in the Journal of Symbolic Logic **46** (1981), 435–436.

—. 1985. *Free intuitionistic logic and its S4 counterpart*, Logique et Analyse **28**, 283–294.

—. 1989. *Programming logics: An introduction to verification and semantics*, New York, Wiley.

—. 1994. *Free arithmetic*, The Journal of Symbolic Logic **59**, 717–718. (abstract.)

—. 1996. *Free logic in program specification and verification*, in E. Morsher (editor), *New directions in free logic* (Bonn, Akademia Verlag).

Theodore HAILPERIN. 1957. *A theory of restricted quantification I*, The Journal of Symbolic Logic **22**, 19–35.

C. A. R. HOARE. 1969. *An axiomatic basis of computer programming*, Communications of the ACM **12**, 576–583.

C. P. J. KOYMANS and G. R. RENARDEL DE LAVALETTE. 1989. *The logic MPL_ω*, M. Wirsig and J. A. Bergstra (editors), *Algebraic methods: Theory, tools, and applications*, Lecture Notes in Computer Science, Vol. 394 (Berlin/ Heidelberg/New York/Tokyo, Springer-Verlag), 247–282.

Jan KUPER. 1994. *Partiality in logic and computation: aspects of undefinedness*, Ph.D. thesis, University of Twente.

Karel LAMBERT. 1963. *Existential import revisited*, Notre Dame Journal of Formal Logic **4**, 288–292.

—. 1981. *On the philosophical foundations of free logic*, Inquiry **24**, 147–203.

—. 1991. (editor). *Philosophical applications of free logic*, Oxford, University Press.

—. 1997. *Nonextensionality*, W. Lenzen (editor), *Festschrift für Franz von Kutschera* (Berlin, de Gruyter, 1997?); forthcoming.

Karel LAMBERT and Bas VAN FRAASSEN. 1972. *Derivation and counterexample: An introduction to philosophical logic*, Encino, CA, Dickenson.

Hugues LEBLANC. 1976. *Truth-value semantics*, Amsterdam, North-Holland.

—. 1982. *Existence, truth, and provability*, Albany, State University of New York Press.

H. LEBLANC and R. D. GUMB (editors). 1983a. *Essays in epistemology and semantics*, New York, Haven.

—. 1983b. *Soundness and completeness proofs for three brands of intuitionistic logic*, in [Leblanc and Gumb *1983a*], 163–197. Abstract in the Journal of Symbolic Logic **46** (1981), 201–202.

J. D. MCLEAN. 1990. Review of [Gumb *1989*], ACM Computing Reviews, August 1990, 405. (9008–0639.)

Karl MENGER. 1979. *A counterpart of Occam's razor*, in his *Selected papers in logic and foundations, dialectics, and economics* (Dordrecht, Reidel), 105–135.

Robert K. MEYER and Karel LAMBERT. 1968. *Universally free logic and standard quantification theory*, The Journal of Symbolic Logic **33**, 8–26.

David L. PARNAS. 1972. *On the criteria to be used in decomposing systems into modules*, Communications of the ACM **15**, 1053–1058.

—. 1993. *Predicate logic for software engineering*, IEEE Transactions on Software Engineering **19**, 856–861.

Willard V. O. QUINE. 1963. *On what there is*, in his *From a logical point of view* (New York, Harper Torchbooks), 1–19.

Wim RUITENBURG. 1991. *The unintented interpretations of intuitionistic logic*, T. Drucker (editor), *Perspectives on the history of mathematical logic* (Boston/Basel/Berlin, Birkhäuser), 134–160.

Rolf SCHOCK. 1965. *On definitions*, Archiv für mathematische Logik und Grundlagenforschung **8**, 28–44.

Joseph SCHOENFIELD. 1967. *Mathematical logic*, Reading, MA, Addison-Wesley.

Dana S. SCOTT. 1967. *Existence and description in formal logic*, R. Schoenmann (editor), *Bertrand Russell, philosopher of the century* (London, Allen and Unwin), 181–200. (Reprinted in [Lambert *1991*].)

—. 1970. *Advice on modal logic*, K. Lambert (editor), *Philosophical problems in logic* (Dordrecht, Reidel), 143–173.

—. 1979. *Identity and existence in intuitionistic logic*, in [Fourman, Mulvey, and Scott *1979*], 660–696.

Raymond M. SMULLYAN. 1968. *First-order logic*, Berlin, Springer-Verlag.

Manfred SCHMIDT-STRAUSS. 1987. *Computational aspects of order-sorted logic with term declarations*, Lecture Notes in Computer Science, Vol. 395, Berlin/Heidelberg/New York/Tokyo, Springer-Verlag.

Patrick SUPPES. 1957. *Introduction to logic*, New York, D. van Nostrand.

Richmond THOMASON. 1969. *Symbolic logic: an introduction*, New York.

—. 1995. *Logicism: Exact philosophy, linguistics, and artificial intelligence*. Talk presented at the Twenty-fifth Annual Meeting of the Society for Exact Philosophy, Calgary, Alberta, May 1995.

Simon **THOMPSON**. 1989. *A logic for Miranda*, Formal Aspects of Computing **1**, 339–365.

A. S. **TROELSTRA** and D. van **DALEN**. 1988. *Constructivism in mathematics: An introduction*, Vol. I, Amsterdam/New York/Oxford/Tokyo, North-Holland.

Wladyslaw M. **TURSKI** and Thomas S. E. **MAIBAUM**. 1987. *The specification of computer programs*, Wokingham, England Addison-Wesley.

Michael **UNTERHALT**. 1986. *Kripke-Semantik für Logik mit partieller Existenz*, Ph.D. thesis, Westfählischen Wilhelms-Universität Münster.

Paulo A. S. **VELOSO**. 1993. *A new, simpler proof of the modularization theorem for logical specifications*, Bulletin of the IGPL **1**, 3–12.