Formalizations Of Substitution Of Equals For Equals

David Gries [*] and Fred B. Schneider [†]
Computer Science Department, Cornell University
Ithaca, New York 14853 USA

May 27, 1998

Abstract. Inference rule "substitution of equals for equals" has been formalized in terms of simple substitution (which performs a replacement even though a free occurrence of a variable is captured), contextual substitution (which prevents such capture), and function application. We show that in connection with pure first-order predicate calculus, the function-application and no-capture versions of the inference rule are the same and are weaker than the capture version. We discuss the deductive apparatus needed for the no-capture version to be as powerful as the capture version.

1 Introduction

About three hundred years ago, Gottfried Wilhelm Leibniz introduced the following characterization of equality (see e.g. [5]):

> Two terms are the same (*eadem*) if one can be substituted for the other without altering the truth of any statement (*salva veritate*). If we have P and Q, and P enters into some true proposition, and the substitution of Q for P wherever it appears results in a new proposition that is likewise true, and if this can be done for every proposition, then P and Q are said to be the *same*; and conversely, if P and Q are the same, they can be substituted for one another as I have said.

In [2], Church proves that the following formulation of one-half of Leibniz's characterization of equality holds in pure predicate calculus F1:

(1) **Substitution of equals for equals:** If S results from R by substitution of Q for P at one or more places in R (not necessarily at all occurrences of P in R), and if $\vdash P \equiv Q$, then $\vdash R \equiv S$.

[*] Supported by NSF grant CDA-9214957.
[†] Supported in part by ARPA/RADC grant F30602-96-1-0317 and AFOSR grant F49620-94-1-0198. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.

Since then, (1) has become a cornerstone of calculational formulations of logic (see e.g. [3, 6]), so named because a formalization of (1) is the central inference rule. However, differing formalizations of (1) have been used. One [3] is in terms of function application, another [7] is in terms of substitution that doesn't avoid capture of free occurrences of variables, and a third is in terms of substitution that does avoid such capture [6]. This article explores the relation between these three formalizations.

In Sect. 2, we present Church's first-order logic F1 [2] and define two different notions of substitution as well as the first formalization of (1), called Leibniz. In Sect. 3, we introduce introduce logic LF, which includes Leibniz as an inference rule, and prove that F1 and LF have the same theorems.

In Sect. 4, we give two other formulations of (1), Leibniz-FA (for Function Application) and Leibniz-CS (for Contextual Substitution), and argue that they are just different notations for the same thing. We then construct logic CSF from LF by replacing Leibniz with Leibniz-CS and prove that some theorems of LF are not theorems of CSF, establishing that, in the context of pure first-order predicate calculus, Leibniz-CS is weaker than Leibniz. In Sect. 5, we describe two ways to extend CSF to give Leibniz-CS (and thus Leibniz-FA) the same power as Leibniz.

2 Church's F1

Terminology from Church [2] will be used. We use p, q, r for propositional variables; v, x, y, z for individual variables; $c, c_1, c_2, \ldots$ for individual variables or constants; f, g for functional variables or constants; and $A, B, C, D, E, P, Q, R, S, V$ for syntactical variables (which denote formulas).

Formulas (i.e. wffs) are formed according to the following rules.

1. A propositional variable is a wff.
2. For f an n-ary functional variable or functional constant and $c_1, \ldots, c_n$ individual variables or individual constants or both (not necessarily all different), $f(c_1, \ldots, c_n)$ is a wff. If f is 1-ary, we depart from Church's notation and write $f.c$ instead of $f(c)$.
3. For P a wff, $\neg P$ is a wff.
4. For P and Q wffs, $(P \Rightarrow Q)$ is a wff.
5. For P a wff and x an individual variable, $(\forall x)P$ is a wff. Variable x is the *dummy* of $(\forall x)P$. We abbreviate $(\forall x)P$ by $(x)P$ (as does Church [2]).

An occurrence of individual variable x is *bound in formula* P iff the occurrence is within a subformula of P of the form $(x)Q$; otherwise, the occurrence of x is *free in* P.

Precedence conventions allow the elimination of some parentheses. Our order of precedence, with the highest first, is: substitution operators (see below); the prefix operators negation $\neg$ and quantification (x); $\vee$ and $\wedge$; $\Rightarrow$ and $\Leftarrow$; and $\equiv$ and $\not\equiv$.

We use the following abbreviations:

Consequence: $P \Leftarrow Q$ **is** $Q \Rightarrow P$
Disjunction: $P \vee Q$ **is** $(P \Rightarrow Q) \Rightarrow Q$
Conjunction: $P \wedge Q$ **is** $\neg(\neg P \vee \neg Q)$
Equivalence: $P \equiv Q$ **is** $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
Inequivalence: $P \not\equiv Q$ **is** $\neg(P \equiv Q)$
Truth: $true$ **is** $p \equiv p$
Falsity: $false$ **is** $\neg true$

An abbreviation can always be replaced by the formula that it abbreviates, and vice versa, e.g. $(x)(f.x \vee p)$ can be replaced by $(x)((f.x \Rightarrow p) \Rightarrow p)$. Such a replacement is done at the "metalevel". That is, in logic F1, to be introduced later, $(x)(f.x \vee p)$ and $(x)((f.x \Rightarrow p) \Rightarrow p)$ are the same formula, so that theorem $(x)(f.x \vee p) \equiv (x)((f.x \Rightarrow p) \Rightarrow p)$ of F1 is an instance of Reflexivity of $\equiv$, $P \equiv P$. Frequently, the distinction between the logic and the metalevel is blurred, but in this paper it is necessary to recognize the distinction.

In formalizing (1), we need a notation for expressing the substitution of Q for P in a formula R. For this purpose, we introduce the notion of a *placeholder* for a formula. A *formula scheme* R^α is a formula R in which some its subformulas may have been denoted by placeholder α. For example, R^α could be $(P \vee \alpha) \wedge Q$. The notation R^α_S then denotes formula scheme R^α with α replaced by S. For example, $((P \vee \alpha) \wedge Q)^\alpha_S$ is $(P \vee S) \wedge Q$, and $((x)(\alpha \vee P))^\alpha_{f.x}$ is $(x)(f.x \vee P)$.

Note that a formula scheme R^α is not a formula; it becomes a formula only when all occurrences of its placeholder are replaced by a formula, as in R^α_P.

Using a placeholder, we can formalize Substitution of equals for equals (1) as the following inference rule [1]:

$$\vdash P \equiv Q \quad \longrightarrow \vdash R^\alpha_P \equiv R^\alpha_Q$$

We use the term *contextual substitution* to describe the replacement of a propositional variable in a formula by a formula in a way that avoids capture of free variables.

(2) **Contextual substitution**: For p a propositional variable and P, Q formulas, $P[p := Q]$ denotes a copy of P in which all occurrences of p are replaced by Q, but with dummies renamed to avoid capture. We define $P[p := Q]$ recursively as follows.

[1] An inference rule $\vdash P_1, \ldots, \vdash P_n \quad \longrightarrow \vdash Q$ means: if P_1, ..., P_n are theorems (of the logic under consideration), then so is Q. We also write such an inference rule as $\frac{P_1, \ldots, P_n}{Q}$.

$p[p := Q]$ **is** Q
$q[p := Q]$ **is** q for q a propositional constant or variable different from p
$c[p := Q]$ **is** c for c an individual variable or constant
$(f(c_1, \ldots, c_n))[p := Q]$ **is** $f(c_1, \ldots, c_n)$
$(\neg P)[p := Q]$ **is** $\neg(P[p := Q])$
$(P \Rightarrow R)[p := Q]$ **is** $P[p := Q] \Rightarrow R[p := Q]$
$((z)P)[p := Q]$ **is** $(z)(P[p := Q])$ if z does not occur free in Q
$((z)P)[p := Q]$ **is** $((y)(P[z := y]))[p := Q]$ if z occurs free in Q,
where y does not occur free in P or Q

Church [2] presents the following first-order predicate logic F1, consisting of two inference rules and five axioms. Inference rule (3) and axioms (4)–(6) form the propositional part of the logic. Since the axioms contain syntactical variables, they are really axiom schemes, which stand for all the axioms constructed by replacing the syntactical variables by formulas. In the same fashion, proofs of theorems in F1 are really proofs of theorem schemes.

(3) **Modus Ponens:** $\vdash P \Rightarrow Q, \vdash P \longrightarrow \vdash Q$

(4) **Axiom Affirmation of the Consequent:** $P \Rightarrow (Q \Rightarrow P)$

(5) **Axiom $\Rightarrow$ over $\Rightarrow$:** $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))$

(6) **Axiom Contrapositive:** $(\neg P \Rightarrow \neg Q) \Rightarrow (Q \Rightarrow P)$

(7) **Generalization:** $\vdash P \longrightarrow \vdash (x)P$

(8) **Axiom $\Rightarrow$ over $\forall$:** $(x)(P \Rightarrow Q) \Rightarrow (P \Rightarrow (x)Q)$
for x an individual variable that doesn't occur free in P

(9) **Axiom Instantiation:** $(x)P \Rightarrow P[x := c]$
for x an individual variable and c an individual variable or constant

3 Calculational logic LF

Below, we present calculational logic LF, with propositional part (10)–(23). LF contains Leibniz (10) as its formalization of (1). Logic LF has more axioms than F1 because LF does not rely on abbreviations to introduce operators beyond $\Rightarrow$.

(10) **Leibniz:** $\vdash P \equiv Q \longrightarrow \vdash R^{\alpha}_{P} \equiv R^{\alpha}_{Q}$

(11) **Transitivity of $\equiv$:** $\vdash P \equiv Q, \vdash Q \equiv R \longrightarrow \vdash P \equiv R$

(12) **Equanimity:** $\vdash P, \vdash P \equiv Q \longrightarrow \vdash Q$

(13) **Axiom Associativity of $\equiv$:** $((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))$

(14) **Axiom Symmetry of $\equiv$:** $P \equiv Q \equiv Q \equiv P$

(15) **Axiom Identity of $\equiv$:** $\mathit{true} \equiv P \equiv P$

(16) **Axiom $\neg$ over $\equiv$:** $\neg(P \equiv Q) \equiv \neg P \equiv Q$

(17) **Axiom Associativity of $\vee$:** $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$

(18) **Axiom Symmetry of $\vee$:** $P \vee Q \equiv Q \vee P$

(19) **Axiom Idempotency of $\vee$:** $P \vee P \equiv P$

(20) **Axiom $\vee$ over $\equiv$:** $P \vee (Q \equiv R) \equiv P \vee Q \equiv P \vee R$

(21) **Axiom Excluded middle:** $P \vee \neg P \equiv \mathit{true}$

(22) **Axiom Golden rule:** $P \wedge Q \equiv P \equiv Q \equiv P \vee Q$

(23) **Axiom Implication:** $P \Rightarrow Q \equiv P \vee Q \equiv Q$

(24) **Generalization:** $\vdash P \longrightarrow \vdash (x)P$

(25) **Axiom $\Rightarrow$ over $\forall$:** $(x)(f.x \vee \neg f.x \Rightarrow (P \Rightarrow Q)) \Rightarrow (P \Rightarrow (x)(f.x \vee \neg f.x \Rightarrow Q))$
for x an individual variable that doesn't occur free in P

(26) **Axiom Instantiation:** $(x)(f.x \vee \neg f.x \Rightarrow P) \Rightarrow P[x := c]$
for an individual variable x and individual variable or constant c,
where P does not contain a free occurrence of f

Axioms $\Rightarrow$ over $\forall$ (25) and Instantiation (26) are obtained from axioms of the same names in logic F1 by replacing subformulas of the form $(x)R$ by $(x)(f.x \vee \neg f.x \Rightarrow R)$. The change to these more complicated axioms is what allows us to prove, later on, that Leibniz-CS (30) is weaker than Leibniz (10).

Logic LF is designed to support a calculational style of proof. To illustrate, in Fig. 1 we prove $\vdash_{\mathrm{LF}}$ (9) $\equiv$ (26), i.e. we prove that (27) is a theorem of LF.

(27) $(x)P \Rightarrow P[x := c] \quad \equiv \quad (x)(f.x \vee \neg f.x \Rightarrow P) \Rightarrow P[x := c]$.

Each of the two equivalences in the proof, broken across three lines, is a consequence of an instance of inference rule Leibniz (10), and the premise of that inference rule is given as an indented hint on the middle line (containing $\equiv$). For example, the first three lines prove $(\forall x)(f.x \vee \neg f.x \Rightarrow P) \Rightarrow P[x := c] \equiv (\forall x)(\mathit{true} \Rightarrow P) \Rightarrow P[x := c]$ because it is the conclusion of an instance of Leibniz (10) whose premise is Excluded middle (21). Transitivity of $\equiv$ (11) is used to conclude that the first and last formulas in this proof

$$
\begin{array}{ll}
 & (x)(f.x \vee \neg f.x \Rightarrow P) \Rightarrow P[x := c] \\
\equiv & \quad \langle \text{Excluded middle (21)} \rangle \\
 & (x)(\mathit{true} \Rightarrow P) \Rightarrow P[x := c] \\
\equiv & \quad \langle\, \mathit{true} \Rightarrow Q \equiv Q \ \text{(a theorem of LF)} \rangle \\
 & (x)P \Rightarrow P[x := c]
\end{array}
$$

Figure 1: Proof of $\vdash_{\mathrm{LF}}$ (9) $\equiv$ (26)

format are equivalent. In the proof, we use Symmetry of $\equiv$ without explicit mention.

(28) **Theorem.** F1 and LF have the same theorems.

Proof. We show that the axioms of one are theorems of the other and the inference rules of one are (perhaps derived) inference rules of the other.

Church [2] introduces a propositional logic P1 and proves that P1 and the propositional part of F1 (i.e. (3)–(6)) have the same theorems and (derived) inference rules. In [7] [2], we prove that the propositional part of logic LF (i.e. (10)–(23)) has the same theorems and (derived) inference rules as P1. Hence, we conclude that the propositional parts of F1 and LF have the same (derived) inference rules and theorems.

Now for the non-propositional parts of F1 ((7)–(9)) and LF ((24)–(26)). First, Generalization is an inference rule of both logics.

Second, consider Instantiation axioms (9) and (26). The proof given above proves $\vdash_{\mathrm{LF}}$ $(9) \equiv (26)$. Since the inference rules and theorems used in the proof are (derived) inference rules and theorems of F1, also $\vdash_{\mathrm{F1}}$ $(9) \equiv (26)$. By Equanimity (12) —a (derived) inference rule of both logics— (9) and (26) are theorems of both logics.

Finally, in the same fashion, we can prove that the axioms named $\Rightarrow$ over $\forall$ in F1 and LF are theorems of F1 and LF. □

4 Calculational logic CSF

We now give two other formalizations of Substitution of equals for equals (1). Leibniz-FA (29), introduced and used by Dijkstra and Scholten in [3] [3], is in terms of function application. Leibniz-FA asserts that two applications (of the same function) with equivalent arguments give equivalent results.

(29) **Leibniz-FA** : $\vdash P \;\equiv\; Q \quad\longrightarrow\quad \vdash f.P \;\equiv\; f.Q \quad .$

Note that $f.P$ is not a formula of F1 or LF (arguments of f must be individual variables or constants), so we look for another way to express this inference rule. We can view a function as a lambda expression $(\lambda p.E)$. Function application $(\lambda p.E).P$ is conventionally defined to have the same value as the formula $E[p := P]$ —this definition is called β -conversion (see e.g. [1]). Hence, Leibniz-FA (29) can be rewritten as:

(30) **Leibniz-CS** : $\vdash P \;\equiv\; Q \quad\longrightarrow\quad \vdash E[p := P] \;\equiv\; E[p := Q] \quad .$

[2] The calculational propositional logic in [7] is not exactly the same as the calculational part of LF, but it is readily seen to be equivalent to it.

[3] Actually, monograph [3] presents the formula $[P \equiv Q] \Rightarrow [f.P \equiv f.Q]$, where "$[E]$" has the interpretation "E is everywhere true", but then uses this implication as if it were an inference rule.

Moreover, any formula E can be abstracted to a function $(\lambda p.E)$ of one of its propositional variables, so we could write $E[p := P]$ as the function application $(\lambda p.E).P$. Hence, we can view Leibniz-FA (29) and Leibniz-CS (30) as equivalent.

Note that Leibniz-FA (29) is an instance of Leibniz (10) —Leibniz-FA is Leibniz with formula R replaced by $f.\alpha$. Hence Leibniz is at least as powerful as Leibniz-FA (and thus Leibniz-CS (30)). We want to prove that, in the context of pure first-order logic, Leibniz is indeed more powerful than Leibniz-FA and Leibniz-CS.

To that end, consider a new logic CSF, which is LF with Leibniz (10) replaced by Leibniz-CS (30). Observe that the proof of (27) on p. 5 is not a CSF-proof because the first substitution-step,

$$\begin{array}{ll} & (x)(f.x \vee \neg f.x \;\Rightarrow\; P) \;\Rightarrow\; P[x := y] \\ \equiv & \quad \langle \text{Excluded middle (21)} \rangle \qquad\qquad\qquad \text{(Step is incorrect in CSF)} \\ & (x)(\mathit{true} \;\Rightarrow\; P) \;\Rightarrow\; P[x := y] \end{array}$$

is incorrect. Why? Well, this step is supposed to follow from inference rule

$$\frac{f.x \;\vee\; \neg f.x \;\equiv\; \mathit{true}}{(x)(f.x \vee \neg f.x \;\Rightarrow\; P) \;\Rightarrow\; P[x := y] \;\;\equiv\;\; (x)(\mathit{true} \;\Rightarrow\; P) \;\Rightarrow\; P[x := y]}$$

which, written as an instance of Leibniz-CS, would have to be something like

$$\frac{f.x \;\vee\; \neg f.x \;\equiv\; \mathit{true}}{E[p := f.x \vee \neg f.x] \;\equiv\; E[p := \mathit{true}]}$$

where E is $(x)(p \Rightarrow P) \Rightarrow P[x := y]$. But the contextual substitution $E[p := f.x \vee \neg f.x]$ does not yield the desired result because dummy x must be changed before the contextual substitution can be carried out.

Moreover, one is led to suspect that Instantiation (9) is not even a theorem of CSF for the following reason. In CSF, all quantifications in the axioms have the form $(x)(f.x \vee \neg f.x \Rightarrow P)$, and inference rule Leibniz-CS cannot be used to change $f.x \vee \neg f.x$ in the body of such quantifications without changing dummy x first. We can indeed prove that Instantiation (9) is not a theorem of CSF. Our proof rests on the following theorem.

(31) **Theorem.** Suppose formula scheme S^α doesn't contain "$(x)f.x$" for an individual variable x. If $\vdash_{\mathrm{CSF}} S^\alpha_{(x)f.x}$ then $\vdash_{\mathrm{CSF}} S^\alpha_V$.

Proof. We prove the theorem by induction on the length of the proof of a formula $S^\alpha_{(x)f.x}$.

Base case. For a proof that requires 0 inference steps, $S^\alpha_{(x)f.x}$ is an instance of one of the axioms (13)–(23), (25)–(26). We prove this case for axiom (26); the rest are similar.

$S^\alpha_{(x)f.x}$ is $(x)(f.x \vee \neg f.x \Rightarrow P) \Rightarrow P[x := c]$ for some formula P. Observe that each occurrence of the subformula $(x)f.x$ in this formula lies entirely within P.[4] Hence, we

[4] This observation is critical. The base case would not hold if, for example, $(x)P \Rightarrow P[x := c]$ were an axiom of CSF, because replacing P by $f.x$ would yield $(x)f.x \Rightarrow (f.x)[x := c]$ and $V \Rightarrow (f.x)[x := c]$ is not valid.

can write $S^\alpha_{(x)f.x}$ as $(x)(f.x \vee \neg f.x \Rightarrow P^\alpha_{(x)f.x}) \Rightarrow P^\alpha_{(x)f.x}[x := c]$, where P^α does not contain $(x)f.x$. But then S^α_V is $(x)(f.x \vee \neg f.x \Rightarrow P^\alpha_V) \Rightarrow P^\alpha_V[x := c]$ and, since this formula is an instance of (26), it is an axiom, so $\vdash_{\mathrm{CSF}} S^\alpha_V$.

Inductive case. Consider an arbitrary formula schema S^α for which a proof of $S^\alpha_{(x)f.x}$ requires at least one inference rule. We investigate the four possibilities for the last inference rule of the proof.

Case Generalization $\vdash P^\alpha_{(x)f.x} \quad \longrightarrow \quad \vdash (y)P^\alpha_{(x)f.x}$**.** Here, S^α is $(y)P^\alpha$. Since $P^\alpha_{(x)f.x}$ is proved earlier in the proof, by the induction hypothesis, $\vdash P^\alpha_V$. Then, by Generalization, $\vdash (y)P^\alpha_V$, i.e. $\vdash S^\alpha_V$.

Case Equanimity $\vdash Q^\alpha_{(x)f.x}, \vdash Q^\alpha_{(x)f.x} \equiv S^\alpha_{(x)f.x} \quad \longrightarrow \quad \vdash S^\alpha_{(x)f.x}$**.** Assume that Q^α does not contain occurrences of $(x)f.x$ (otherwise, replace them by α). Since $Q^\alpha_{(x)f.x}$ and $Q^\alpha_{(x)f.x} \equiv S^\alpha_{(x)f.x}$ are proved earlier in the proof, by the induction hypothesis, $\vdash Q^\alpha_V$ and $\vdash Q^\alpha_V \equiv S^\alpha_V$. Hence, by Equanimity, $\vdash S^\alpha_V$.

Case Transitivity $\vdash P^\alpha_{(x)f.x} \equiv Q^\alpha_{(x)f.x}, \ \vdash Q^\alpha_{(x)f.x} \equiv R^\alpha_{(x)f.x} \quad \longrightarrow \quad \vdash P^\alpha_{(x)f.x} \equiv R^\alpha_{(x)f.x}$**.** Here, S^α is $P^\alpha \equiv R^\alpha$. Assume that Q^α does not contain occurrences of $(x)f.x$ (otherwise, replace them by α). Since $P^\alpha_{(x)f.x} \equiv Q^\alpha_{(x)f.x}$ and $Q^\alpha_{(x)f.x} \equiv R^\alpha_{(x)f.x}$ are proved earlier in the proof, by the induction hypothesis, $\vdash P^\alpha_V \equiv Q^\alpha_V$ and $\vdash Q^\alpha_V \equiv R^\alpha_V$. Hence, by Transitivity, $\vdash P^\alpha_V \equiv R^\alpha_V$, i.e. $\vdash S^\alpha_V$.

Case Leibniz-CS $\vdash P \equiv Q \quad \longrightarrow \quad \vdash E[p := P] \equiv E[p := Q]$**.** Here, $S^\alpha_{(x)f.x}$ is $E[p := P] \equiv E[p := Q]$. We have to determine S^α. Every occurrence of $(x)f.x$ in the conclusion is contained wholly in one of E, P, and Q —e.g. an occurrence could not have been formed by having P be $f.x$ and having E contain $(x)p$, because $((x)p)[p := f.x]$ is $(y)f.x$ and not $(x)f.x$. Therefore, we can rewrite the inference rule being applied in this case as follows. Let E^α, P^α, and Q^α be E, P and Q with every occurrence of $(x)f.x$ replaced by α, so that E^α is $E^\alpha_{(x)f.x}$, and similarly for P and Q. Then, the inference rule is

$$\frac{\vdash P^\alpha_{(x)f.x} \equiv Q^\alpha_{(x)f.x}}{\vdash E^\alpha_{(x)f.x}[p := P^\alpha_{(x)f.x}] \equiv E^\alpha_{(x)f.x}[p := Q^\alpha_{(x)f.x}]}$$

which can be written as:

$$\frac{\vdash P^\alpha_{(x)f.x} \equiv Q^\alpha_{(x)f.x}}{\vdash (E^\alpha[p := P^\alpha] \equiv E^\alpha[p := Q^\alpha])^\alpha_{(x)f.x}}$$

so S^α is $E^\alpha[p := P^\alpha] \equiv E^\alpha[p := Q^\alpha]$. Since $P^\alpha_{(x)f.x} \equiv Q^\alpha_{(x)f.x}$ is proved earlier in the proof, by the induction hypothesis, $\vdash P^\alpha_V \equiv Q^\alpha_V$, so by Leibniz-CS, $\vdash S^\alpha_V$. □

(32) **Corollary.** There is a theorem of LF that is not a theorem of CSF.

Proof. $V \Rightarrow f.y$ is not valid and is thus not a theorem of CSF, so, by the contrapositive of theorem (31), $(x)f.x \Rightarrow f.y$ is not a theorem of CSF. Since $(x)f.x \Rightarrow f.y$ is a theorem of LF, the theorem follows. □

5 Extending CSF

We have shown that Leibniz-CS (30) and its equivalent Leibniz-FA (29) are weaker than Leibniz (10) in the context of a pure predicate logic. With additional apparatus, however, all three rules can have the same power.

Perhaps the simplest way to extend CSF so that Leibniz (10) and Leibniz-FA (29) have the same power is to use a formulation of Leibniz-FA that caters to the replacement of functions, as defined in Church [2, p. 192].[5] Informally, this kind of substitution calls for replacing function applications by the body of the function, with the parameters of the function replaced by the arguments of the application. For a function $f(x)$ with body E, the value of a function application $f.y$ is $E[x := y]$. Thus, Church defines contextual substitution for a function as follows (but using our notation).

Let f be an n-ary functional variable, x be a list $x_1, \ldots, x_n$ of distinct individual variables (the parameters of f), and E be a formula. The notation $P[f(x) := E]$ denotes a copy of P in which each (and every) function application $f.y$ (for y a list $y_1, \ldots, y_n$ of individual variables/constants) is replaced by $E[x := y]$ —with two restrictions that ensure that no capture takes place during the substitution.

1. If E contains a free occurrence of an individual variable y (that is not in parameter list x), then no subformula $(y)S$ of P may contain f.[6]
2. If a function application $f.y$ occurs in P and if E contains a subformula $(y1)Q$ for $y1$ an individual variable in list y, then Q may not contain a free occurrence of a variable in list x.[7]

We replace Leibniz-CS (30) in logic CSF by the following:

(33) **Leibniz-FA′** : $\vdash P \equiv Q \quad \longrightarrow \vdash E[f(x) := P] \equiv E[f(x) := Q] \quad .$

Relying on the theorem $P \equiv f.x \vee \neg f.x \Rightarrow P$, we can prove (9) $\equiv$ (26) using the single instance of (33) given below, where g is chosen so as not to occur in P.

$$\frac{P \;\equiv\; f.x \vee \neg f.x \;\Rightarrow\; P}{((x)g.x \Rightarrow P[x := c])[g(x) := P] \;\equiv\; ((x)(g.x) \Rightarrow P[x := c])[g(x) := f.x \vee \neg f.x \;\Rightarrow\; P]}$$

A second method has the same effect but is more tortuous to use. Admit lambda terms (e.g. $(\lambda p.E)$) as terms within formulas, introduce $=$ as a 2-ary function (written as an infix operator), introduce contextual substitution for individual variables and function variables as well as propositional variables, and generalize all three Leibniz inference rules to allow

[5] We are indebted to Vladimir Lifschitz for this observation.

[6] This restriction avoids the capture that would happen in the substitution $((y)f.z)[f(x) := g.y]$, which would yield $(y)(g.y)$.

[7] This restriction avoids the capture that would happen in the substitution $(f.y)[f(x) := (y)g.x]$, which would yield $(y)(g.y)$.

substitution for equal lambda terms. Then use the following axioms concerning equality of lambda terms as well as conventional β-conversion (below, g is a functional variable, functional constant, or lambda term):

(34) **Extensionality:** $\vdash P \equiv Q \quad \longrightarrow \quad \vdash (\lambda p.P) = (\lambda p.Q)$
$\vdash P \equiv Q \quad \longrightarrow \quad \vdash (\lambda x.P) = (\lambda x.Q)$
$\vdash P \equiv Q \quad \longrightarrow \quad \vdash (\lambda f.P) = (\lambda f.Q)$

(35) β**-conversion:** $(\lambda q.P).Q$ **is** $P[q := Q]$
$(\lambda x.P).c$ **is** $P[x := c]$
$(\lambda f.P).g$ **is** $P[f := g]$

The proof of the following theorem eliminates the need to replace a subformula that contains a bound variable by lifting the subformula to a function application and replacing the function instead [8] .

(36) **Theorem.** Leibniz (10) is a derived inference rule in any extension of CSF that includes Extensionality (34) and β-conversion (35).

Proof. We assume $\vdash P \equiv Q$ and prove $\vdash R^\alpha_P \equiv R^\alpha_Q$ Let f be a fresh functional variable (it does not appear in R, P, or Q), and let v be the list of individual variables that occur free in P or Q. Since $\vdash P \equiv Q$, by Extensionality (34), $\vdash (\lambda v.P) = (\lambda v.Q)$. We calculate:

$$\begin{array}{ll}
 & R^\alpha_P \\
\equiv & \quad \langle \text{Property of contextual substitution: } P[v := v] \text{ is } P \rangle \\
 & R^\alpha_{P[v:=v]} \\
\equiv & \quad \langle \beta\text{-conversion (35)} \rangle \\
 & R^\alpha_{(\lambda v.P).v} \\
\equiv & \quad \langle\ (\lambda v.P) = (\lambda v.Q) \text{ (see text that precedes this calculation)} \\
 & \qquad \text{— } (\lambda v.P) \text{ and } (\lambda v.Q) \text{ contain no free variables} \rangle \\
 & R^\alpha_{(\lambda v.Q).v} \\
\equiv & \quad \langle \beta\text{-conversion (35)} \rangle \\
 & R^\alpha_{Q[v:=v]} \\
\equiv & \quad \langle \text{Property of contextual substitution: } Q[v := v] \text{ is } Q \rangle \\
 & R^\alpha_Q
\end{array}$$

□

The proof given above obscures an important point. The middle step of the proof uses Leibniz-CS (30), but the two steps involving β-conversion are at the meta-level, because β -conversion is an abbreviation. Thus, this proof uses substitution at two levels, and they are different kinds of substitution —Leibniz (10) is used at the metalevel and Leibniz-CS (30) is used in the logic itself.

[8] We are indebted to Rutger Dijkstra [4] for introducing us to this technique.

Suppose we wrote β-conversion as an axiom instead of as an abbreviation:

(37) **Axiom** : $(\lambda p.P).Q \ \equiv \ P[p := Q]$

Now the second step in the proof above could not have been carried out using Leibniz-CS (30) because the subformula being replaced, $P[v := v]$, contains free occurrences of the variables in list v, and some of these might be bound in $R^{\alpha}_{P[v:=v]}$. Thus, our proof of (36) depends critically on β-conversion being an abbreviation instead of an axiom.

6 Discussion

Whether one uses Leibniz (10), Leibniz-FA′ (33), or Leibniz-FA (29) (with the additional apparatus mentioned above) is perhaps a matter of background and taste. We are drawn to Leibniz for the following reasons. First, Leibniz is a far simpler to use than the two alternatives. Second, Leibniz (10) is needed at the metalevel —if it is formalized— for abbreviations. Having Leibniz as an inference rule within the logic lets us do the same kind of substitution at both levels, allowing the distinction between the two levels to recede —as is commonly done anyway.

Acknowledgements

We are indebted in particular to Rutger Dijkstra, who gave us several insightful comments over the years, and to Valdimir Lifschitz for pointing us to the use of Church's substitution for functions. Thanks also to Roland Backhouse and Rick Aaron for useful comments on previous drafts of this paper.

References

[1] Barendregt, H.P. *The Lambda Calculus, its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

[2] Church, A. *Introduction to Mathematical Logic.* Princeton University Press, Princeton, 1956.

[3] Dijkstra, E.W., and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer Verlag, NY, 1990.

[4] Dijkstra, R.M. Private communication, 18 December 1996.

[5] Gerhardt, K.I., (ed.). *Die Philosophishen Schriften von G.W. Leibniz, Band VII, "Scientific Generalis. Characteristica", XIX and XX.* Weidmannshe Buchh., Berlin, 1887.

[6] Gries, D., and F.B. Schneider. *A Logical Approach to Discrete Math.* Springer Verlag, NY, 1993.

[7] Gries, D., and F.B. Schneider. *Equational propositional logic. IPL 53* (1995), 145-152.