A DECISION PROCEDURE FOR PURE FIRST-ORDER LOGIC

TIMM LAMPERT

Humboldt University Berlin, Unter den Linden 6, D-10099 Berlin
e-mail address: lampertt@staff.hu-berlin.de

Abstract. This paper first criticizes Church's and Turing's proofs of the undecidability of first-order logic and explains this critique against the background of a modest version of finitism inspired by Wittgenstein. It identifies assumptions of Church's and Turing's proofs to be rejected, justifies their rejection and argues for a new discussion of the decision problem. The main part of the paper then challenges the widely accepted negative solution to the decision problem by specifying a decision procedure for pure first-order logic without identity (FOL) that the author has implemented in a program called the "FOL-Decider". Thus, a purely logical discussion of the decision problem is made possible. The logical foundations, proof strategies and termination criteria of the FOL-Decider program are explained. It is proven that the FOL-Decider is correct and terminates. Thus, it is proven that FOL is decidable (contrary to the Church-Turing theorem).

Contents

keywords: pure first-order logic, decision problem, Church-Turing theorem, proof search strategy, unification, finitism, Wittgenstein
Mathematics Subject Classification: 03B10

1. Introduction

First-order logic without names, functions or identity (FOL) is the simplest first-order language to which the Church-Turing theorem applies. I have implemented a decision procedure, the FOL-Decider, that is designed to refute this theorem. The purpose of part 1 of this paper is to explain my motivation to do so. Part 2 then explains the implemented decision procedure and proves its validity.[1]

The decision procedure described in this paper requires the application of nothing but well-known rules of FOL. No assumption is made that extends beyond equivalence transformation within FOL. In this respect, the decision procedure is independent of any controversy regarding foundational matters.

However, to explain why it is reasonable to work out such a decision procedure despite the fact that the Church-Turing theorem is widely accepted, it is convenient to begin from a critique of the proofs of the theorem as well as a general overview of the background that motivates my work.

Section 2 relates my critique to Turing's and Church's proofs. Here, I identify the assumption of each proof that I question, and I explain why I question it. My critique is directed against the claim that it is possible to express a decision function for FOL in a language based on FOL. I argue that any undecidability proof for FOL is underdetermined because, rather than reducing the hypothetical decidability assumption to absurdity, one can instead reduce to absurdity the assumption that the translation of such a decision function into a language based on FOL indeed expresses that decision function.

Section 3 then explains my argument against the general background of several significant distinctions drawn by Wittgenstein. According to my interpretation, these distinctions present an alternative understanding of the relation between computation and logic to that implied in Church's or Turing's proof. Against this conceptual background, undecidability proofs of FOL are ruled out. Instead, the decidability of FOL constitutes a natural conjecture that logicians should address.

In section 3, I refer to Wittgenstein's discussion of the foundations of mathematics since my endeavour was originally motivated by my efforts (i) to make his programmatic claims concerning decidability reasonable and (ii) to present a systematic account of his conception of a "new logic" that he placed in opposition to the mathematical logic established by Frege and Russell. However, I do not pretend to offer any exegesis of Wittgenstein in section 3. Instead, I intend to point out some distinctions and views espoused by Wittgenstein that challenge the Church-Turing theorem and that inspired me to work out a decision procedure for FOL. What is at issue is not the interpretation of Wittgenstein but rather the reason for reconsidering the Church-Turing theorem. Similarly, by advocating a modest version of Wittgensteinian finitism in section 3.3, I refer to neither any discussion about Wittgenstein's standpoint on the foundations of mathematics in general nor his relation to finitism in particular,[2] nor do I pretend to do justice to Wittgenstein's way of discussing mathematics and its foundations, which did not aim to provide a systematic account. I also do not make any effort to relate the "modest version of Wittgensteinian finitism"

[1] The FOL-Decider as well as further programs are available at the following link:
http://www2.cms.hu-berlin.de/newlogic/webMathematica/Logic/home.jsp.

[2] Frascolla, Marion and Rodych were the first to relate Wittgenstein's philosophy of mathematics to finitism, and I have profited considerably from their thorough work, e.g., from [Frascolla (1994)], [Marion (1998)] and [Rodych (1999)].

that I advocate to other versions of finitism. My intention is merely to sketch the general framework that has motivated my work on the decision problem.

The Polish philosopher and logician Leon Gumański also criticised the Church-Turing theorem ([Gumański (1988)]) and offered a proof of the decidability of FOL (cf. [Gumański (2000)] and [Gumański (2008)]). His critique is based on a general critique of the diagonal method, including a critique of Cantor's theorem. In contrast, my critique questions only the application of the diagonal method in undecidability proofs of FOL. Unlike Gumański's argument, it addresses the problem of the reliability of the intended interpretations of diagonalized expressions within a language based on FOL. Gumański's proof of the decidability of FOL consists of defining an upper limit for proofs in Beth's tableaux calculus. This proof is questioned by [Matzer (2018)] and [Mycka & Rosa (2018)], who argue that proofs in Beth's calculus that capture the computation of the Ackermann-Péter function (Matzer) or the Goodstein series (Mycka and Rosa) necessarily extend Gumański's primitive recursive upper bound. In contrast, my decision procedure specifies a new proof strategy (the ∧I-minimal proof strategy) and concerns the implementation of syntactic criteria for terminating the proof search. My reasoning for questioning the Church-Turing theorem and my decision procedure are inspired by Wittgenstein and are independent of Gumański.

The main part of this paper, part 2, explains the decision procedure implemented in the FOL-Decider. My decision procedure is based on a correct and complete calculus that I call the "NNF-calculus". In the NNF-calculus, the rule of ∧-introduction, ∧I, plays a prominent role. It is applied iteratively in a search for a proof within the NNF-calculus. Its iterative application would not terminate in any arbitrary case if the proof search were not restricted such that all proof paths ultimately enter a "detectable loop" in the case of unprovability. The FOL-Decider implements a "∧I-minimal proof strategy" to ensure that a systematic proof search will either find a proof with a minimum number of ∧I applications or will ultimately terminate all proof paths due to what I call the "Loop Criterion".

Modern logic engines, such as Vampire, i-prover or Spass, manage to decide many satisfiable formulas (infinity axioms included) by the method of saturation based on a resolution calculus. This method refines the search for proofs by preserving the correctness and completeness of the presumed calculus. My decision procedure can be conceived as an alternative to known saturation algorithms, which are based on resolution.

Part 1. **Reconsidering the Decision Problem**

2. Critique of the Church-Turing Theorem

Undecidability proofs of FOL rest on proofs concerning the expressibility of computable functions within a language based on FOL. It is argued that a contradiction follows from expressing a decision function for FOL within a language based on FOL in the case of diagonalization. This contradiction is used to reduce the assumption that FOL is decidable to absurdity. This section argues that this conclusion is not conclusive because the possibility to express a decision function for FOL within a language based on FOL cannot be proven without assuming that FOL is undecidable. Thus, the contradiction may likewise be used to reduce the assumption to absurdity that a decision function for FOL is expressible within a language based on FOL. By analysing proofs based on the diagonal method as proofs of undefinability theorems, it is argued that this alternative conclusion is straightforward.

I introduce the terminology needed for this analysis in sections 2.1 to 2.3. Section 2.1 is related to expressing computable functions by Turing machines or recursive functions. Section 2.2 discusses the idea of expressing propositional functions by open formulas of a language based on FOL. Section 2.3 then analyses proofs by contradiction based on the diagonal method as inexpressibility theorems. Based on this analysis, I will analyse Turing's proof (section 2.4) and Church's proof (section 2.5) and argue that they can be analysed as inexpressibility proofs of a decision function for FOL within a language based on FOL instead of inexpressibility proofs of a decision function for FOL in a language of computation such as Turing machines or recursive functions.

2.1. **Expressing a Computable Function.** Undecidability proofs can be analysed as proofs that show that certain functions cannot be expressed by Turing-computable or recursive functions. This section specifies the notion of expressing a computable function.

The general problem in undecidability proofs is the question of whether certain functions are isomorphic.

Definition 2.1. Two functions are *isomorphic* iff

(1) their domains can be correlated in a one-to-one manner,
(2) their ranges can be correlated in a one-to-one manner, and
(3) all and only correlated values are assigned to correlated arguments.

Definition 2.2. Functions can be *mapped onto each other* iff they are isomorphic.

Definition 2.3. An *interpretation of a function* f is a set of rules for mapping a function g onto f.

Definition 2.4. A function f *expresses* a function g iff f is interpreted such that g is mapped onto f.

Remark 2.5. Typically, proofs that show that a function g cannot be expressed by functions f of a specific type are proofs by contradiction. For the sake of reducing an assumption to absurdity, one may assume that a function f expresses a function g although, in fact, f cannot express g. In that case, there is no interpretation of f that maps g onto f. Although no such interpretation exists, one '*intends* to interpret' f such that it can be mapped onto g. The proof then shows that such an intention must fail: no set of rules can be specified to realize this intention. Thus, the *intended interpretation* of a function f of a specific type is not '*reliable*'. I will specify this terminology in relation to the interpretation of FOL expressions below. However, I will also apply it to functions, as explained in this remark.

I presume familiarity with the concepts of Turing machines and recursive functions. String functions computed by Turing machines provide a paradigm for computable functions. The question of the computability of a function g in which we are interested can be posed as the question whether string functions f can be interpreted such that g can be

mapped onto f.[3] *Turing's thesis* states that any computable function can be expressed by a Turing-computable string function.

Definition 2.6. *Turing's thesis*: A function is computable iff it is expressible by a Turing-computable string function.

Church's thesis specifies an alternative statement based on number-theoretic recursive functions.

Definition 2.7. *Church's thesis*: A function is computable iff it is expressible by a recursive function.

Since one can specify rules for mapping Turing-computable string functions and recursive functions onto each other, these two theses are provably equivalent.

Definitions 2.3 to 2.7 suffer from the vagueness of the *concept of a rule* in Definition 2.3. However, there is no need to enter a discussion of this concept in this paper. By referring to rules, I wish to exclude non-standard mappings such as a mapping of natural numbers to a binary notation for natural numbers such that all and only binary strings ending with '0' are strings of numbers of halting Turing machines. Such a notation would make the halting problem decidable (cf. [Boker & Dershowitz (2008)], p. 206, applying [Shapiro (1982)], p. 17, corollary C2a). Such a non-standard mapping cannot be specified by any rules. It is merely specified by a general description that says nothing about *how* to realize the mapping. Therefore, the halting problem is not computable according to the above definitions, even though there may exist such a non-standard mapping relative to a non-standard notation that is not available through any interpretation in terms of a set of rules.

I also leave it an open question whether the concept of a rule implies the concept of computability, which would render the Church-Turing thesis circular. This is a well-known problem in the literature (cf., in particular, [Rescorla (2007)]) that this paper does not address. The same applies to the general problem of giving a precise definition of the Church-Turing thesis that specifies *in general, precisely and non-circularly* how to express arbitrary functions by Turing-computable string functions or recursive functions. This paper is not concerned with a critique of the Church-Turing thesis. Instead, I presume the standard terminological background for undecidability proofs. I consider the (im)possibility of deciding FOL in a standard way under the assumption that no problems arise from the application of the Church-Turing thesis. It is uncontroversial that any program written in an ordinary computer language can be mapped onto a Turing-computable string function by applying rules of a concrete translation procedure that can itself be carried out by a computer program. Thus, the question at stake can be specified in a very concrete way, independently of the notorious problems with giving a precise *general* definition of a computable function: Do undecidability proofs indeed prove that the endeavour to specify a decision procedure for FOL-provability in an ordinary computer language is futile?

[3] Cf., e.g., [Weihrauch (2000)], p. 3:

> Ordinary computability theory first introduces computable partial word functions $f\colon \subseteq \Sigma^* \to \Sigma^*$ explicitly, for example, by means of Turing machines. For defining computability on other sets M (rational numbers, finite graphs, etc.) words are used as 'names' or 'codes' of elements of M. While a machine still transfers words to words, the user interprets these words as names of elements from the set M.

2.2. **Expressing a Propositional Function.** Computer languages in general and Turing machines or definitions of recursive functions in particular are languages for expressing computable functions. In contrast, FOL is, first and foremost, a language that can be used to express propositional functions by open formulas containing at least one free variable relative to an interpretation $\Im$. Interpretations $\Im$ of FOL expressions obey the *principles of extensional semantics*: a propositional variable is interpreted by a proposition that refers to one and only one truth value (principle of bivalence), a variable is interpreted by a set serving as the domain of quantification, a name symbol is interpreted by one and only one object of the domain, an n-place function symbol is interpreted by a function mapping n-tuples of the domain to values of the domain, an n-place predicate symbol is interpreted by a set of n-tuples from the domain, and logical connectives and quantifiers are interpreted as truth functions mapping atomic propositional functions such that the truth value of a complex proposition depends on nothing but the referential relations of its constituents.

Definition 2.8. An *interpretation $\Im$ of an FOL expression φ* assigns extensions (truth values, objects, or sets) to the respective constituents of φ in accordance with the principles of extensional semantics.

Definition 2.9. A *number-theoretical interpretation of an FOL expression φ* interprets φ with the natural numbers as the domain of quantification.

Definition 2.10. The language of arithmetic, L_A, is based on FOL, with '=' as the only predicate and with the addition of the constant '0', the one-place function symbol 'S', and the two-place function symbols '+' and '×'. L_A is interpreted by its standard fixed number-theoretical semantics $\Im_A$.

While interpretations $\Im$ of FOL expressions are not fixed and are not necessarily number-theoretical in nature, the interpretation $\Im_A$ of L_A is fixed and number-theoretical. '0' is interpreted to refer to the number 0; 'S' is interpreted by the successor functions; '+', by addition; '×', by the multiplication of natural numbers; and '=', by equality. I call this the 'standard fixed number-theoretical semantics of L_A'.

Definition 2.11. The *interpretation $\Im_A$ of an L_A expression φ* assigns extensions (truth values, natural numbers, or sets of n-tuples of natural numbers) to the respective constituents of φ in accordance with the standard fixed number-theoretical semantics of L_A.

L_A is 'a language based on FOL', i.e., translation rules can be defined that eliminate all the vocabulary extending FOL such that logical properties, e.g., satisfiability, are preserved. In the following, I abstain from considering languages other than L_A that are based on FOL.[4]

Definition 2.12. A *propositional function $f(x)$ is expressed by an open L_A- or FOL formula* $\varphi(x)$ relative to an interpretation $\Im$ iff for all x that satisfy $f(x)$, the objects of the domain $\Im(x)$ satisfy $\Im(\varphi)$.

This definition can be trivially extended to n-place propositional functions. Henceforth, I tacitly presume that considerations concerning one-place functions also extend to n-place functions.

Definition 2.13. An *intended interpretation I* of an L_A or FOL expression φ assigns ordinary expressions to the constituents of φ with the intention of providing an interpretation $\Im(\varphi)$.

[4]Definition 2.12 and the terminological distinction between expressing and capturing (cf. below Definition 2.21 are reminiscent of [Smith (2013)], p. 41.

Example 2.14. The intended interpretations of the vocabulary with fixed interpretations are $I(\neg)$ = 'not', $I(\wedge)$ = 'and', $I(\vee)$ = 'or', $I(\rightarrow)$ = 'if ... then ...', $I(\leftrightarrow)$ = 'if and only if', $I(\forall)$ = 'all', $I(\exists)$ = 'some', $I(0)$ = 'zero', $I(S)$ = 'successor of', $I(+)$ = 'plus', and $I(\times)$ = 'multiplied by'.

Remark 2.15. L_A or FOL formulas can be paraphrased as ordinary propositions through the use of intended interpretations.

Unlike interpretations ($\Im$), intended interpretations (I) do not obey the principles of extensional semantics *by definition*. Rather, intended interpretations can be *unreliable* (inadmissible, not permissible, or impossible).

Definition 2.16. An intended interpretation I of an L_A or FOL expression φ is *reliable* iff it provides an interpretation $\Im(\varphi)$.

This definition does not provide any *criterion* for identifying unreliable intended interpretations. The identification of (un)reliable interpretations is a non-trivial problem.[5] An unreliable intended interpretation may *intend* to provide a reliable interpretation of on open formula in terms of a (well-defined) propositional function. However, this intention may fail to be realized. The mere assignment of well-formed ordinary propositions to well-formed FOL expressions does not guarantee that the intended interpretations, in fact, yield well-defined propositional functions.

FOL is *sound* relative to interpretations $\Im$ but not to intended interpretations I since the latter can be unreliable.[6] One does not conclude that FOL is unsound because interpreting the FOL proof $\exists x(Fax \wedge Gx) \vdash \exists x Gx$ by 'Jack seeks something that is a unicorn; therefore, something is a unicorn' yields an inference that is not truth-preserving (here making use of the following intended interpretations: $I(x)$ = the set of all beings, $I(F)$ = '... seeks ...',

[5]For example, [Peregrin and Svoboda(2017)], p. 70, propose an inferential 'criterion for reliability (REL)' that compares formal and informal inferential relations (cf., e.g., [Brun(2003)] for another extensive discussion of the topic). One might use REL to identify *unreliable* intended interpretations in the case that informal judgements are available; if, e.g., $I(\varphi)$ does not follow informally from $I(\psi)$ (i.e., the inference is not judged to be truth-preserving) but $\psi \vdash \varphi$, then $I(\varphi)$ and $I(\psi)$ cannot both be reliable. Of course, informal judgements can be assumed neither to be reliable nor to be available. This applies, in particular, to the intended interpretations of the hypothetical diagonal cases involved in undecidability proofs (cf. the following sections). However, criteria such as REL show that one cannot simply infer the validity of intended interpretations from the provability of the formulas that are interpreted. Independent evidence is needed. In the case of undecidability proofs, such independent evidence may well reside in the specification of a decision procedure, which would demonstrate the unreliability of the intended interpretations involved.

[6][Kreisel (1967)], pp. 152-154, deliberately distinguishes 'intuitive logical validity' (= logical validity according to an intended interpretation I) from 'truth in all set-theoretic structures' (= truth in all $\Im$), stating merely that the former implies the latter without *assuming* that the latter also implies the former. However, the assumption that provability implies 'intuitive logical validity' is assumed by Kreisel without any further argument, and this assumption is not deliberate ([Smith (2013)], p. 356, likewise confuses soundness with 'evident validity'). Kreisel uses this assumption as the basis of his erroneous 'informal rigorous proof' for the equivalence of provability, 'intuitive logical validity' and 'truth in all set-theoretic structures' given the equivalence of provability in a sound and complete calculus and truth in all set-theoretic structures. However, the soundness of FOL calculi is related to extensional semantics ('set-theoretic structures', in Kreisel's words), and there is no way to infer the truth of instances or intended interpretations of provable formulas from the soundness of FOL inference rules. One would instead need to show that the intended interpretations obey extensional semantics. However, there is no rigorous way to do this; discussions of non-extensional, so-called 'opaque' contexts are endless in the philosophy of logic. Typically, they are identified by arguing that instances or intended interpretations of *provable* FOL formulas cannot be judged to be true.

$I(a)$ = 'Jack', and $I(G)$ = '... is a unicorn'). Instead, one concludes that 'seeks' is not a predicate that obeys the principles of extensional semantics since its second position is not 'referential'. It may well be that intended interpretations of provable formulas φ of a language based on FOL are not true; in such a case, the intended interpretations are not reliable.

Example 2.17. $I(P)$ = 'This proposition is not true' is not reliable since it does not refer unambiguously to a truth value, according to the Liar Paradox. One cannot infer from the provable formula $P \leftrightarrow P$ that 'This proposition is not true iff This proposition is not true' is true.

Example 2.18. $I(F(x, y))$ = 'x is a member of y' and $I(x, y)$ = 'the set of all sets' are not reliable since these interpretations do not refer unambiguously to sets (Russell's Paradox). One cannot infer from the provability of $\neg\exists x\forall y(Fyx \leftrightarrow \neg Fyy)$ that the following statement is true: 'Not: Something is a set x of all sets y such that y is a member of x iff y is not a member of y' (more fluently, 'A set composed of all normal sets does not exist').

Remark 2.19. The analysis of paradoxes is controversial. Ramsey's distinction between semantic paradoxes (e.g., the Liar Paradox) and logical or set-theoretical paradoxes (e.g., Russell's Paradox) is prominent (but also controversial). Ramsey claimed that only the former show that certain expressions do not validly refer (or have a well-defined meaning), although one might think that they do, while the latter are veridical paradoxes proving that certain axioms or assumptions that one might think to be true are, in fact, false. According to this analysis, Russell's Paradox falsifies the unrestricted comprehension axiom schema of naive set theory (i.e., the schema $\exists x\forall y(y \in x \leftrightarrow \varphi(y))$. Such an analysis presumes that the intended interpretations of logical formulas are reliable in the case of logical or set-theoretical paradoxes. However, the analysis given in Example 2.18 contradicts this analysis; according to the analysis given in Example 2.18, the analysis of Russell's Paradox as a veridical paradox commits the *fallacy of intended interpretations*, as defined in Definition 2.20. However, Example 2.18 is given only for the sake of illustration. The reasoning put forward in this paper presumes neither any *analysis of paradoxes* nor any *general claim* concerning the possibility of expressing 'self-reference' within the language of logic. It is merely assumed that intended interpretations may violate the semantic principles of FOL (for whatever reason) and, thus, it does not go without saying that one can infer the truth of intended interpretations of provable formulas. It is the analysis of logical or set-theoretical paradoxes as veridical paradoxes that risks sanctioning such an inference on the basis of Ramsey's controversial *general claim* concerning his distinction of paradoxes. This paper argues that any proof resting on such an inference should be justified by independent reasoning and that a critical analysis of established proofs must scrutinize whether such a reasoning is, in fact, provided.

Definition 2.20. The *fallacy of intended interpretations* is the inference of the truth of $I(\varphi)$ from the provability of φ.

Committing this fallacy is suggestive since (i) interpreting logical formulas by intended interpretations seems to sanction those interpretations as logical (= obeying the laws of logic and its semantics) and (ii) intended interpretations are intended to refer to set-theoretical structures, and it comes as a paradoxical, counterintuitive surprise when one realizes by some independent reasoning that they do not, in fact, refer to that to which they are

intended to refer. Semantic intention or intuition with regard to reference relations is not a criterion for successful reference, however.

For the following, it is sufficient to acknowledge that intended interpretation may not be reliable and that, therefore, one cannot infer the truth of an intended interpretation from the provability of the formula it interprets. Independent evidence is needed to argue for the reliability of intended interpretations. This is especially true when diagonalization is involved (cf. the following section).

2.3. **Inexpressibility Theorems and the Diagonal Method.** The diagonal method can be analysed as a method of showing that a certain function cannot be expressed by a certain type of function. This section presents the details of this analysis.

The term 'diagonalization' stems from Cantor's application of the diagonal method by referring to the *diagonal of a matrix*. Cantor defines an *anti*-diagonal by a function g that assigns to each position n a value that differs from that in position n of the n-th entry of the matrix. This method of defining an anti-diagonal is used to prove that no function f defining a sequence of values in the matrix expresses g. To prove this, it is assumed that g would define a function enumerated in the matrix. One can refer to the enumerated entries in a matrix by indices. The diagonal method shows that g cannot be identified with some function f_n defining the values in the n-th row since the value in the n-th position of the sequence defined by f_n would not be well defined in that case. Diagonalization is either the process or the result of taking the index of a function f_n as its own argument. The diagonal method is the application of diagonalization to prove that no function f_n can express a function g defined over the enumeration of functions f since it is impossible to interpret any of the enumerated f_n as g.

A so-called *non-matrix-topological* definition can be formulated by defining the diagonalization of *open formulas* $\varphi(x)$ in a language based on FOL. Let $\ulcorner\gamma\urcorner$ be the numeral of the Gödel number of $\varphi(x)$; then, the process of replacing x with $\ulcorner\gamma\urcorner$—or the result of this process, $\varphi(\ulcorner\gamma\urcorner)$—is the diagonalization of $\varphi(x)$.

Undecidability proofs based on the diagonal method typically involve the application of the *Diagonalization Lemma*. This lemma presumes an axiom system that is at least as strong as the (very weak) Robinson Arithmetic Q of elementary arithmetic. Q can be formulated in L_A. Although it is not proven rigorously in an absolute sense that Q is consistent, Q is assumed to be sound relative to $\Im_A$ (and, consequently, to also be consistent). Though one may question this assumption from a finitistic point of view, I take the soundness and consistency of Q for granted. In particular, Q is strong enough to *capture* diagonalization in terms of a recursive function *diag*. While the notation for expressing a function refers to the interpretation of a language (e.g., $\Im_A$ in the case of L_A), the notion of capturing refers to provability within a theory (cf. [Smith (2013)], p. 119):

Definition 2.21. A theory T *captures* the one-place numerical function f by means of the open wff $\varphi(x, y)$ iff, for any m, n,

(i) if $f(m) = n$, then $T \vdash \varphi(\overline{m}, \overline{n})$, and
(ii) $T \vdash \exists! y \varphi(\overline{m}, y)$.

Here, (ii) satisfies the uniqueness condition for functions. Note that (i) and (ii) imply that if $f(m) \neq n$, then $T \vdash \neg\varphi(\overline{m}, \overline{n})$.

Diagonalization in terms of a recursive function *diag* is independent of any intended interpretation of the result of diagonalization. It is based on the purely syntactic action of

replacing a variable in a formula with a certain Gödel numeral, which, in turn, is defined through purely syntactic means. I do not question that diagonalization in this sense is definable in terms of a recursive function *diag*, nor that this recursive function can be expressed by a function *Diag* within L_A and captured by any theory T that is at least as strong as Q. Referring to *Diag* in the diagonalized function $\varphi(x)$ makes it possible to derive the *Diagonalization Lemma*:

Definition 2.22. If T is at least as strong as Q and $\varphi(x)$ is an open formula with one free variable x defined in a language at least as rich as L_A, then there exists a closed formula γ in the language of T such that $T \vdash \gamma \leftrightarrow \phi(\ulcorner\gamma\urcorner)$.

I take this lemma (and, consequently, its proof) for granted. This lemma concerns the provability of a formula, not the validity of its intended interpretations.

Many paradoxes, such as the Liar Paradox, Russell's Paradox and Richards' Paradox, are based on the diagonalization of *open sentences of either ordinary language or a semi-formal language*. One can diagonalize 'x is not true' by replacing 'x' in 'x is not true' with an expression (such as 'this proposition') that is intended to refer to the resulting expression. Similarly, 'x is a set that is not a member of itself' is diagonalized by replacing 'x' with an expression that is intended to refer to a set specified by 'x is a set that is not a member of itself'.

Here, I will define diagonalization for *expressions* of the form $f(x)$, such as $f_n(x)$, open formulas $\varphi(x)$ or open sentences such as 'x is not true'. To do so, I use the general term 'code' for any sort of sign or denotation of such an expression, be it an index, a Gödel numeral, a deictic expression or some other sort of sign. I do not presume that such a code indeed refers to the expression to which it is assigned (although it is intended to refer to it). For example, one can generate the Gödel numeral $\ulcorner\gamma\urcorner$ for a formula φ, and this numeral can occur in a formula ψ; whether $\ulcorner\gamma\urcorner$ refers to a number, to φ, or to both according to a reliable interpretation of ψ is not presumed in the process of generating $\ulcorner\gamma\urcorner$. Furthermore, I refer to the *result* of the process of replacing x in $f(x)$ with the code for $f(x)$ as the 'diagonalization'.

Definition 2.23. Let $\ulcorner\gamma\urcorner$ be the code for $f(x)$; then, $f(\ulcorner\gamma\urcorner)$ is the *diagonalization* of $f(x)$.

This definition is rather general; it does not imply that $\ulcorner\gamma\urcorner$ is the code for $f(\ulcorner\gamma\urcorner)$. According to the *Diagonalization Lemma*, for example, it is only by diagonalizing an open formula $\varphi(x)$ containing *Diag* that a resulting formula $\varphi(\ulcorner\gamma\urcorner)$ is obtained that contains an expression $Diag(\ulcorner\gamma\urcorner)$ that is provably equivalent to the Gödel number of $\varphi(\ulcorner\gamma\urcorner)$.

The diagonal method is based on diagonalization. In contrast to diagonalization, the diagonal method possesses the two features (i) and (ii) identified in Definition 2.24 *by definition* (cf. [Jacquette (2004)], p. 70):

Definition 2.24. The *diagonal method* is a method of deriving proofs by contradiction by making use of diagonalization, hence implying (i) some sort of inversion, denial, negation or imposition of the complement of a diagonalization and (ii) some sort of self-application (or intended self-reference) in the diagonalization resulting from (i).

Definition 2.25. A *diagonal argument* is an argument based on the diagonal method.

Definition 2.26. A *diagonal case* is a diagonalization used via the diagonal method in a proof by contradiction.

Definition 2.24 is admittedly vague. Any definition of the diagonal method depends on the extent of the proofs one intends to cover. For the following discussion of Turing's and Church's proofs, a general and rather vague characterization of the diagonal method that is consistent with its standard use suffices. Neither a discussion of the diagonal method in general nor its general validity is at stake here; only the application of this method in proofs of the undecidability of FOL is of interest.

One may question Definition 2.24 not only because it is vague but also because its reference to 'proofs by contradiction' is either too narrow or inadequate. The former critique can be justified by, e.g., the distinction between good and bad diagonal arguments (cf. [Simmons (1993)], p. 29): Bad diagonal arguments do not reduce to absurdity the assumption of an entity that is not well defined, whereas good diagonal arguments either do not assume any entity that is not well defined or reduce to absurdity the assumption that some entity is well defined (which is a standard analysis to which I refer). The latter critique of the classification of diagonal arguments as proofs by contradiction may be justified by noting that proofs by contradiction are a specific class of logical proofs that presume well-defined entities (cf., e.g., RFM IV, §28, as well as the intuitionistic project of a negationless mathematics as formulated by [Griss (1946)] and [Nakano (2019)] for relating Wittgenstein's and Griss' analyses of proofs by contradiction in mathematics). I do not question these critiques. I simply follow the traditional understanding of the application of the diagonal method in undecidability proofs such as Turing's and Church's. As the analyses in sections 2.4 and 2.5 will show, it is standard to analyse these proofs as proofs by contradiction that reduce to absurdity the assumption that it is possible to define a decision function for FOL-provability by applying the diagonal method as characterized by the features specified in Definition 2.24. I discuss neither the relation of these proofs to other proofs using the diagonal method nor the question of whether it is adequate to formalize diagonal arguments within a language based on FOL.

It is, however, important to note that diagonalization is defined for *expressions*, not functions. The intention of this distinction is to explicate the application of the diagonal method to prove inexpressibility (or so-called 'undefinability') theorems, which is crucial to the following discussion. One cannot assume that intended interpretations of an expression of the form $f(x)$ express a function if those intended interpretations are intended to apply to diagonal cases. Even if the intended interpretation of expressions of the form $f(x)$ is a function, one cannot assume that this intended interpretation is also reliable in diagonal cases. The intention to interpret a function f_n as, for example, g (defining the *anti*-diagonal over an enumeration of functions $f_1, f_2, \ldots$) fails due to its diagonalization: it does not make sense to speak of a resulting diagonalized function since the result does not satisfy the requirement that a function must assign one and only one value to each of its arguments. Therefore, expressions of the form $f(x)$ do not necessarily express a function; only their reliable interpretations do. The function g is well defined only as a function that is not part of the enumeration over which it is defined. However, there is no interpretation that maps g onto a function f_n due to diagonalization. The diagonal method proves the impossibility of the intention to express a function in a certain way. Similarly, 'x is not true' and 'x is not a member of itself' are not well-defined propositional functions if their diagonalizations are intended to be interpreted as self-referential propositions.

For the discussion of the Church-Turing theorem, the extent to which one believes in the possibility of reliable interpretations of diagonalized expressions is irrelevant. No general claim about the reliability of intended interpretations involving 'self-reference' is assumed.

It is sufficient to acknowledge that the diagonal method is used to establish inexpressibility theorems in terms of proofs by contradiction. Before turning to the discussion of Turing's and Church's proofs, I will illustrate the application of the diagonal method to prove inexpressibility theorems based on (i) the impossibility of expressing the halting function by means of Turing machines and (ii) the impossibility of expressing arithmetic truth within L_A.

The impossibility of solving the halting function is an example of an inexpressibility theorem concerning a language for expressing computable functions. It can be proven that the halting function $h(x)$ (which assigns a value of '1' to codes x for Turing machines that halt when started with x and a value of '2' to codes x for Turing machines that do not halt when started with x) *is not expressible by any Turing-computable function.* To show this, the assumption that some Turing machine H can be interpreted as computing the function $h(x)$ is reduced to absurdity by considering a diagonal case. One way to consider such a diagonal case is by considering a composition of a copy machine C, the assumed halting machine H and a dithering machine D that halts iff it is not started with '1' (cf. [Boolos et al. (2003)], pp. 39f.). The dithering machine causes the result of H to be '*inverted*' (feature (i) of Definition 2.24). The diagonal case arises from supposing that CHD is started with its own number (feature (ii) of Definition 2.24). H cannot be interpreted as intended in this diagonal case: if it assigns a value of '1', D makes it such that CHD does not halt, and if it does not assign a value of '1', D makes it such that CHD halts. What is proven is the impossibility of an interpretation: $h(x)$ is well defined only so long as it is not assumed to be computable since otherwise it would be capable of being diagonalized, which is incompatible with its definition as a function. We cannot specify rules for a consistent interpretation of $h(x)$ as a Turing-computable function.

The proof of Tarski's theorem is an example of an inexpressibility theorem concerning a language for expressing propositional functions. It proves that the supposed propositional function 'x is the Gödel number of an L_A formula that is true according to $\Im_A$' (= $\mathrm{True}_{L_A}(x)$) is *not expressible by an open formula in* L_A (for the following, cf., e.g., [Smith (2007)], section 21.5). The assumption that $\mathrm{True}_{L_A}(x)$ is expressible in L_A by an open formula $T_A(x)$ yields a contradiction in the case of the diagonalization of its *negation* $\neg T_A(x)$ (cf. feature (i) in Definition 2.24). Let the Gödel numeral of $\neg T_A(x)$ be $\ulcorner L \urcorner$; then, by the *Diagonalization Lemma*, $L \leftrightarrow \neg T_A(\ulcorner L \urcorner)$ is provable in Q. If Q is sound (as assumed; see p. 9 above), then $L \leftrightarrow \neg T_A(\ulcorner L \urcorner)$ is true according to $\Im_A$. However, if $T_A(x)$ expresses $\mathrm{True}_{L_A}(x)$, then $T_A(\ulcorner L \urcorner) \leftrightarrow L$ is also true according to L_A (cf. feature (ii) in Definition 2.24), which yields a contradiction. Given the *Diagonalization Lemma* and the soundness of Q, this proof reduces to absurdity the assumption that $\mathrm{True}_{L_A}(x)$ is expressible by $T_A(x)$ within L_A. The diagonal method shows that any intended interpretation of an open L_A formula $\varphi(x)$ expressing $\mathrm{True}_{L_a}(A)$ is unreliable. Hence, 'truth in L_A' is a well-defined propositional function only so long as it is not the intended interpretation of an open L_A formula.

These proofs show how the diagonal method is used to prove that certain functions are not expressible either within a language for expressing computable functions (= a language of computation based on Turing machines or recursive functions) or within a language for expressing propositional functions (= a logical language based on FOL). Undecidability proofs for FOL combine both types of languages and consider a characteristic function for FOL-provability. By the diagonal method, it is proven that the assumption that this characteristic function is expressible in both the language of computation *and* the language

of logic yields a contradiction. *Prima facie*, this contradiction can be used only to reduce the *conjunction* to absurdity. However, it is instead used to reduce the first conjunct, the assumption of the decidability of FOL, to absurdity. The reason for reducing the first conjunct to absurdity is the proof of a lemma (Turing) or a theorem (Church) that, roughly speaking, states that *any* computable function is expressible within a language based on FOL. This *general* claim justifies the *specification* to the assumed decision function $Prov(x)$ for FOL-provability and its expressibility in the form of a propositional function $P(x)$: if FOL were decidable, $Prov(x)$ would be expressible as a propositional function $P(x)$ even in the case of the diagonalization of the propositional function $\neg P(x)$. The truth of this claim must be assumed to reduce the first (instead of the second) conjunction of the above conjunct to absurdity. In the following sections, we will take a closer look at the proofs of the corresponding lemma (in Turing's proof) and theorem (in Church's proof) and ask whether these proofs indeed justify this specification.

2.4. **Turing's Proof.** Two sorts of undecidability proofs can be distinguished: *primary* undecidability proofs, which are not based on any other undecidability proof, and *secondary* undecidability proofs, which are based on other undecidability proofs. The undecidability of the halting problem is a primary undecidability proof: it directly refers to a diagonal case to show that it is impossible to interpret any Turing-computable string function as the halting function. Turing's undecidability proof of FOL, in contrast, is a secondary undecidability proof. His undecidability proof rests on the relation of some undecidable function concerning Turing machines (i.e., the halting function) to a logical function of interest (i.e., a decision function for FOL-provability).

To relate properties of Turing machines to properties of formulas, one must interpret those formulas. To describe the instructions and configurations of Turing machines M, an 'intended interpretation' for the non-logical vocabulary in logical formulas is established. For example, [Turing (1936)], pp. 259f., presents the following intended interpretations for the atomic open formulas ('function variables', in his terminology) that he uses:

> $R_{S_l}(x, y)$ is to be interpreted as 'in the complete configuration x (of M), the symbol on the square y is S'.
> $I(x, y)$ is to be interpreted as 'in the complete configuration x, the square y is scanned'.
> $K_{q_m}(x)$ is to be interpreted as 'in the complete configuration x, the m-configuration is q_m'.
> $F(x, y)$ is to be interpreted as 'y is the immediate successor of x'.

By 'complete configuration', Turing refers to what now is called the 'state' of a Turing machine M, while 'm-configuration' is Turing's term for 'configuration'. The details of Turing's intended interpretation, however, are not relevant to the following critique. What is important is that Turing's proof, as well as modern versions of it (cf., e.g., [Boolos et al. (2003)], p. 127), is based on intended (or 'standard') interpretations using ordinary language with an intended meaning. These interpretations are intended to map properties of Turing machines to properties of logical formulas.

As Turing originally defined them, Turing machines start with an empty tape, and 'bad', circular machines are those that become stuck, while good, 'non-circular machines' never halt. In contrast, Turing machines as they have been defined since Post and Davis start with a non-empty tape, and the 'good' ones are those that halt. These differences also

play no essential role in the following discussion. Thus, I will neglect to distinguish between these different concepts of Turing machines.

All versions of Turing's proof relate decision problems regarding the behaviour of Turing machines to the decision problem regarding properties of FOL formulas. It does not matter which logical properties (such as FOL-provability, satisfiability or logical validity) are considered since they are all interdefinable. This is why I discuss simply 'the decidability of FOL', meaning that some logical property of FOL is decidable. Likewise, 'the decision problem' is short for 'the problem of whether some logical property is decidable'.

According to decision problems regarding the behaviour of Turing machines, any function concerning some non-trivial property of Turing machines can be chosen since they are all undecidable according to Rice's Theorem. Turing, for example, refers to the problem of deciding whether a Turing machine ever prints the symbol '0'. Modern versions of Turing's proof intend to map the halting function onto an assumed decision function for FOL formulas. Given such a mapping, the undecidability of FOL follows from the undecidability of the corresponding decision problem for Turing machines.

To relate an undecidable problem concerning the behaviour of Turing machines to the decision problem, a *translation procedure* is defined to translate the codes and configurations of Turing machines into logical formulas (including some background theory). Such a procedure is a purely syntactic exercise and is itself a computable string function. In the following, I refer simply to 'translations of Turing machines M', including the translations concerning their configurations (e.g., the statement that M reaches a halting state at some time), and following Turing, I abbreviate the corresponding formulas by $Un(M)$. Given M and its initial configuration, $Un(M)$ can be computed. The question is whether a one-to-one correlation can be established between the logical properties of $Un(M)$ and the properties of M due to the specified translation procedure.

Proving this one-to-one correlation is the crucial element of (all versions of) Turing's proof. This is done by proving a *lemma* that purports to prove that $Un(M)$ is provable (alternatively: true in all interpretations) iff M has the property in question, e.g., ever printing '0' or halting, (= *Turing's Lemma*). The direction from right to left is proven by relating each step of the behaviour of a Turing machine to an inferential, truth-preserving step in a proof of $Un(M)$ (= *Lemma 1*). I will relate my critique only to the direction from left to right (= *Lemma 2*). The proof of this direction is very short and 'easy' ([Boolos et al. (2003)], p. 130). It is based on an intended interpretation of logical formulas describing the configurations of the translated Turing machine. I quote the complete proof of [Turing (1936)], p. 262:

> Lemma 2. If $Un(M)$ is provable, then S_1 [i.e., the symbol '0', T.L.] appears on the tape in some complete configuration of M.
>
> If we substitute any propositional functions for function variables in a provable formula, we obtain a true proposition. In particular, if we substitute the meanings tabulated on pp. 259-260 in $Un(M)$, we obtain a true proposition with the meaning 'S_1 appears somewhere on the tape in some complete configuration of M'.

Function variables is Turing's term for open formulas, and *propositional functions* is his term for instances (or intended interpretations) of open FOL formulas. The fact that he uses the term 'propositional functions' shows that he does not question that any instance or intended interpretation is, in fact, reliable. If one could assume that any instance or intended interpretation of an open FOL formula is indeed a (well-defined) propositional

function, then he would be quite correct that any such instance or intended interpretation of a provable formula would be true. However, this cannot be presumed, as shown by the discussion in section 2.2 and the discussion of Tarski's theorem. As his second sentences makes evident, Turing, in fact, refers to intended interpretations of predicates used in $Un(M)$ (quoted above on p. 13), and he stipulates in the first sentence of his proof that from the provability of a formula, the truth of its intended interpretation follows. Hence, his 'proof' is a nice example of the fallacy of intended interpretations (cf. Definition 2.20).

Turing's intended interpretation of $Un(M)$ seems to be unproblematic as long as applied to normal (non-diagonal) cases. One might argue that, unlike in the case of paradoxes, the question of reliability does not arise because Turing machines and their description are well determined. However, this view poses a risk of succumbing to the fallacy of intended interpretation. The question is whether the intended interpretations of $Un(M)$ are also reliable in the diagonal cases that are relevant to the primary undecidability proofs on which Turing's proof is based. It was argued in section 2.2 that the unreliability of intended interpretations may come as a surprise, contradicting semantic intuitions, and it was argued in section 2.3 that the diagonal method is used to demonstrate that intended interpretations do not, in fact, provide the interpretations $\Im$ that they are intended to provide. No effective translation procedures for clear-cut concepts such as Turing machines nor the intuitive evidence of the correctness of the translation in normal cases suffices to demonstrate the correctness of the translation procedure in diagonal cases. Unreliable intended interpretations may well result from completely determined descriptions. Consider, for example, a book consisting of well-formed sentences that are all interpreted as propositions capable of being false or true. The sentences in such a book can be enumerated. Let the book contain the sentence 'Sentence number 27 is not true', and suppose that this sentence itself is the 27th sentence of the book. The sentences in the book, their enumeration, sentences of the form 'Sentence number n is not true' and the intended interpretation of provable formulas of the form $P \leftrightarrow P$ by means of sentences of the form 'Sentence number n is not true' are all well determined, yet they still produce the Liar Paradox. More importantly, one cannot conclude from the unproblematic referential relations of expressions established for typical contexts that they still refer in the same way in atypical contexts such as intensional, metalanguage or tautological contexts. The question is whether diagonal contexts with intended self-referential interpretations are, in fact, extensional contexts or whether they also should be classified as atypical, non-extensional contexts that do not permit inference from the provability of formulas to the truth of their intended interpretations. Neither Turing nor any modern version of his proof addresses the question of the reliability of the intended interpretations in the relevant diagonal cases. For example, [Boolos et al. (2003)], p. 130, presume without further argument that 'truth in all interpretations' ($\Im$) also applies to their 'standard interpretation' (I), without considering the application of their standard interpretation to diagonal cases.

However, if one considers FOL to be decidable, then diagonalization inevitably comes into play as soon as a decision on logical properties is correlated with a decision on the properties of Turing machines. According to the *Turing thesis*, the decidability of FOL implies the existence of a Turing machine that computes a function that expresses a decision function for FOL formulas. Let this Turing machine M be denoted by 'FOL' (in italics). A primary undecidability proof, e.g., a proof of the undecidability of the halting problem, directly applies to the hypothetical machine FOL if *Turing's Lemma*, applied to the halting problem, is assumed.

Let T be a translation machine that generates $Un(M)$ (or its code) from two sequences of strokes m and n: the first is a code for the input of the machine M, and the second codes the instructions for M. According to *Turing's Lemma*, the composition $TFOL$ of T and FOL would behave as a halting machine H since the decision on the provability of $Un(M)$ coincides with the decision on the halting of M with code n when started with m. Applying the diagonal method, as in the case of the proof of the unsolvability of the halting problem, we consider the composition $CTFOLD$ of a copy machine C, a translation machine T, the decision machine FOL and the dithering machine D. This yields a contradiction in the diagonal case: the decision of FOL cannot be interpreted as a decision about the property of the halting of $CTFOLD$ when started with its own number.

This contradiction can be interpreted in two ways. Under the assumption of *Turing's Lemma*, one can reduce to absurdity the assumption that FOL exists (since C, T, and D obviously exist). However, one may likewise reduce the *lemma* to absurdity and argue that the diagonal case of $CTFOLD$ when started with its own number shows that the intended interpretation is not reliable in this special case. In the first interpretation, one questions the expressibility of a decision function for FOL-provability by means of an assumed Turing machine FOL. In the second interpretation, one questions the expressibility of the halting function for Turing machines M by a propositional function $Un(M)$ in the language of FOL. In this interpretation, one would 'withdraw the intended interpretation' of the decision on $Un(M)$ in a diagonal case such as $Un(CTFOLD)$ when started with its own number (cf. RFM, p. 51e, §10). This is a straightforward conclusion since the proof of the *lemma* is based on a fallacy. Thus, what is absurd according to this second interpretation is not the assumption that FOL is decidable but instead the assumption that a decision on $Un(M)$ could be used to decide on the halting function (or some other uncomputable function).

Therefore, instead of rejecting the decidability of FOL by espousing the validity of the mapping of functions concerning the behaviour of Turing machines onto an assumed decision function for FOL by means of intended interpretations, one may reject the possibility of expressing the behaviour of Turing machines by FOL expressions when diagonal cases are involved. This rejection simply applies the proof of the impossibility of expressing the halting function by a Turing-computable function to an assumed decision function for FOL-provability: it is not the computability of the functions onto which the halting function is intended to be mapped that is called into question but rather the claim that such a mapping is even possible in diagonal cases.

An undecidability proof of FOL must be based on a proof of a *lemma* that is independent of FOL's decidability. Versions of Turing's proof do not satisfy this condition since they are based on intended interpretations of $Un(M)$ and these intended interpretations cannot obey the semantics of FOL in the case that FOL is decidable. Therefore, Turing's proof shows only that either FOL is undecidable or properties of Turing machines cannot be expressed by properties of $Un(M)$. Thus, the proof is underdetermined.

The translation procedure for generating formulas $Un(M)$ applies to Turing machines *in general*. Turing's proof assumes that this translation procedure is *correct in any case*, meaning that the resulting intended interpretation of $Un(M)$ does, in fact, express the behaviour of the described Turing machine in any arbitrary case. This *general claim* justifies the argument in the proof of *Lemma 2*. No special reasoning is provided that this general claim, in fact, specifically applies to diagonal cases involving a machine FOL. However, diagonal cases are designed to reduce correlations between functions to absurdity. Therefore, special reasoning is needed to argue that a *lemma* that correlates a function concerning the

behaviour of Turing machines to a function deciding on a logical property of $Un(M)$ holds not merely in normal cases but also in diagonal cases involving FOL.

In such cases, a hypothetical decision function for FOL is assumed to be expressed within FOL. Rejecting the intended interpretation of $Un(M)$ in diagonal cases implies rejecting that FOL-provability is expressible within the language of FOL. This can be seen from the diagonal case of $CTFOLD$ started with its own number. In this case, FOL, which is hypothesized to compute the characteristic function for FOL-provability, is started with $Un(CTFOLD)$, which implies the translation of FOL. This diagonal case makes it impossible to simultaneously trust both what the supposed machine FOL, in fact, decides and what it decides according to the intended interpretation of $Un(CTFOLD)$. If FOL decides that $Un(CTFOLD)$ is provable, it, in fact, assigns the value '1' to $Un(CTFOLD)$; however, since D halts only if not started with '1', it must assign the value '2' (or its code in stroke notation) to $Un(CTFOLD)$ according to the intended interpretation of $Un(CTFOLD)$, which states that $CTFOLD$ halts when started with its own number. Thus, reducing the correctness of the intended interpretation to absurdity (instead of reducing the assumption of the existence of FOL to absurdity) implies rejecting the claim that it is possible to express a decision function for FOL-provability within FOL. A decision function for FOL-provability cannot be expressed within FOL since in the diagonal case, the intended interpretation of such a formula implies a statement on the provability of the very formula that is reduced to absurdity by the diagonal method since it necessarily contradicts the result of the supposed decision machine in the diagonal case.

Turing's proof is only indirectly concerned with the expression of FOL-provability within FOL since it is a secondary undecidability proof that is related to some undecidable property of Turing machines. By contrast, the following discussion of Church's proof will be directly concerned with the question of expressing provability within a language based on FOL.

2.5. **Church's Proof.** While Turing's proof is based on the concept of Turing machines, Church's proof is based on recursive functions. Turing's proof rests on the undecidability of a decision problem for Turing machines that is proven to be unsolvable independent of the technique for expressing properties of Turing machines by logical formulas. Church's proof, in contrast, is not similarly based on the undecidability of a decision problem for recursive functions that can be proven independent of the technique for expressing recursive functions by logical formulas.

Nevertheless, the key feature of both proofs is that the undecidability proof of FOL rests on a technique for expressing computable functions within a language based on FOL. While Turing's proof rests on a translation procedure for expressing the behaviour of Turing machines through logical formulas, Church's proof rests on expressing recursive functions by L_A formulas (which are reducible to FOL formulas). I will argue that the proof of the 'theorem' that every recursive function is expressible within L_A suffers from a similar deficiency as the proof of *Turing's Lemma*.

For simplicity, I refer to modern versions of Church's proof, in particular, to [Smith (2013)].

Church's undecidability proof of FOL rests on a proof that the property of provability in Robinson Arithmetic (Q) is undecidable (= undecidability of Q). Church argues that if FOL is decidable, then Q is decidable. The proof of this implication is based on (i) the translation of L_A formulas into FOL formulas while preserving satisfiability and (ii) the deduction theorem applied to the axioms of Q. The translation rules that eliminate the function symbols (including '0') and identity are defined by [Boolos et al. (2003)], chapter

19.4. These authors prove the correctness of the translation procedure by proving their propositions 19.12 and 19.13 (cf. [Boolos et al. (2003)], pp. 256 and 257). The proofs of these propositions are plain and unproblematic.

One might classify Church's undecidability proof as a secondary proof since it is based on the undecidability of provability in Q. However, this analogy to the classification of Turing's proof as a secondary proof is misleading with regard to the critique of Turing's proof in the previous section. While the reduction of the decision problem to a decision problem for Turing machines is problematic, the reduction of the undecidability of FOL to the undecidability of Q is, in contrast, rather trivial. It does not concern the relation of a language based on FOL to a language of computable functions (in Church's case, recursive functions) but rather concerns the relation of a language based on FOL (in Church's case, L_A) to FOL. If FOL is decidable, then indeed, Q is also decidable. Instead it is the expressibility of recursive functions in L_A, on which the undecidability proof of Q is based, that involves a problem similar to that of Turing's proof. The undecidability proof of Q directly concerns the possibility of expressing an assumed recursive function for Q-provability in L_A and capturing it in Q. Since L_A is reducible to FOL and since provability in Q is reducible to provability in FOL, this proof therefore concerns the possibility of expressing FOL-provability within FOL.

Similar to Turing's proof, which is based on a procedure for translating Turing machines and their configurations into FOL formulas, Church's proof is based on a procedure for translating (canonically) defined recursive functions into L_A formulas. The resulting open L_A formulas are of the rather simple Σ_1 type (i.e., a prenex normal form with only unbounded existential quantifiers). Recursive properties can be defined by means of their characteristic functions. Thus, given a recursive property $f(x)$, one can generate a Σ_1 expression that *expresses* and *captures* the recursive property, given that the translation procedure indeed achieves the purpose for which it is designed.

I call $\neg P(\ulcorner\gamma\urcorner)$ the diagonal case of $P(x)$, where $P(x)$ denotes the supposed Σ_1-type propositional function that expresses and captures a presumed recursive property $Prov(x)$ defined by a characteristic recursive function that is isomorphic to a decision function for Q-provability (implying a decision function for FOL-provability).

According to the *Church thesis*, the assumption that FOL (and, consequently, Q) is decidable implies that a decision function for FOL (and, consequently Q) can be mapped onto a recursive function. As in Turing's proof, the assumption that FOL is decidable is unproblematic as long as it is not related to any diagonal case. In Church's proof, diagonalization comes into the play only when (i) expressing recursive functions within the language of L_A and (ii) considering diagonalization within L_A.

Given the *Diagonalization Lemma* and the consistency of Q (cf. Definition 2.22 and p. 9), the diagonal case immediately shows that $Prov(x)$ cannot exist *if* $Prov(x)$ *is to be expressed and, moreover, captured by* $P(x)$. This conclusion follows trivially from the following three assumptions (cf. [Smith (2013)], p. 183):

Assumption 1: $Q \vdash \gamma \leftrightarrow \neg P(\ulcorner\gamma\urcorner)$.
Assumption 2: If $Q \vdash \gamma$, i.e., $Prov(\ulcorner\gamma\urcorner)$, then $Q \vdash P(\ulcorner\gamma\urcorner)$.
Assumption 3: If $Q \nvdash \gamma$, i.e., not-$Prov(\ulcorner\gamma\urcorner)$, then $Q \vdash \neg P(\ulcorner\gamma\urcorner)$.

Assumption 1 applies the *Diagonalization Lemma* to $\neg P(x)$, Assumptions 2 and 3 follow from the assumption that $P(x)$ captures the presumed recursive property $Prov(x)$. Assumptions 1 to 3 together imply a contradiction. Thus, if Q is consistent and the *Diagonalization*

Lemma holds, then $P(x)$ cannot capture a presumed recursive property $Prov(x)$. A similar argument can be presented on the basis that Q is sound and $P(x)$ expresses $Prov(x)$.

The derived contradiction either can reduce to absurdity the assumption that FOL-provability and, consequently, Q-provability is recursively definable or can reduce to absurdity the claim that the presumed recursive property $Prov(x)$ is expressed and captured by the open formula $P(x)$ resulting from the translation procedure. In the latter case, Church's proof is interpreted not as an undecidability proof but as analogous to the proofs of undefinability theorems, such as the proof of Tarski's theorem. Church, however, advocates for the first option on the basis of the general theorems that (i) any recursive property is expressible within L_A (the *expressing theorem*) and (ii) any recursive property can be captured within Q (the *capturing theorem*). The latter theorem is based on the former. I focus on the former since it is this theorem and its proof that can be questioned in a manner similar to the critique of *Turing's Lemma*.

The procedure for translating the canonical definitions of recursive functions into L_A formulas is itself computable. The crucial question is whether this translation procedure is *correct* in the sense that it indeed results in formulas that *express* the translated recursive functions and properties. The proof of the *expressing theorem* does not prove this rigorously. Instead, it consists of specifying a translation procedure and simply appealing to semantic evidence to argue for its correctness. Smith's proof nicely illustrates this.

The proof of the *expressing theorem* for recursive functions proves the theorem for the initial functions, for the composition of recursive functions, for primitive recursion and for minimization (cf. [Smith (2013)], chapter 15 and p. 297). The question of the criterion for the correctness of the translation procedure already arises in the simplest case of the translations of the initial functions. Smith's proof of the *expressing theorem* starts with proving the theorem for these functions. I quote the complete proof for the successor and zero functions from [Smith (2013)], p. 110:

> *Proof for (1)* There are three easy cases to consider:
>
> i. The successor function $Sx = y$ is expressed by the open wff $\mathtt{Sx} = \mathtt{y}$.
>
> ii. The zero function $Z(x) = 0$ is expressed by the wff $\mathtt{Z(x, y)} =_{\mathtt{def}} (\mathtt{x} = \mathtt{x} \wedge \mathtt{y} = \mathtt{0})$.

In contrast to the tricky case of expressing recursion, the translation procedure is 'trivial' ([Smith (2007)], p. 110) in the case of the initial functions. In this case, its correctness seems to be plain and unquestionable. However, it is interesting to note that the proof, in fact, consists of prescribing *how* the functions are to be translated without explicitly stating *why* the translation is *correct*. The answer to this question must relate the recursively defined functions to the *arithmetic* interpretation $\Im_A$ of the propositional functions in L_A. $\Im_A$ assigns truth values to propositional functions for given instances of their arguments. The translation of a recursively defined function $f(x) = y$ into a propositional function $\phi(x, y)$ is correct iff for all m and n, $\phi(\overline{m}, \overline{n})$ is true according to $\Im_A$ iff $f(m) = n$ (cf. [Smith (2013)], p. 43). How can one know that this is the case?

To answer this question, one must answer the following questions: (i) How can one gain knowledge of the truth values of $\phi(x, y)$ for specific values m and n? (ii) How can one know that the truth values of $\Im_A(\phi(\overline{m}, \overline{n}))$ correspond to the computation of $f(m) = n$? (iii) How can one know the answers for *all* of the infinite number of possible values?

With regard to (i), one cannot refer to provability within Q to justify the truth value of $\Im_A(\phi(\overline{m}, \overline{n}))$ since Q is justified by its soundness and the *capturing theorem* is based on the *expressing theorem*: the syntax is measured by the presumed semantics, not the other way round. So, one must simply accept the truth values of L_A formulas according to $\Im_A$

Recursive Def.	L_A **Formalization**
$0! = 1$ $(Sx)! = x! \times Sx$	$\exists_c \exists_d (\exists_u (c = u \times S(d \times S(0)) + S(0) \wedge$ $\exists_m m + u = c \wedge \exists_m m + S(0) = d \times S(0)) \wedge$ $\exists_u (c = u \times S(d \times S(x)) + y \wedge \exists_m m + u = c \wedge$ $\exists_m m + y = d \times S(x)) \wedge$ $\forall_z (\exists_m m + z = x \Rightarrow (z \neq x \Rightarrow \exists_v \exists_w (w = v \times S(z) \wedge$ $\exists_u (c = u \times S(d \times S(z)) + v \wedge \exists_m m + v = d \times S(z) \wedge$ $\exists_m m + u = c) \wedge \exists_u (c = u \times S(d \times S(S(z))) + w \wedge$ $\exists_m m + w = d \times S(S(z)) \wedge \exists_m m + u = c)))))$

Table 1: Translation of the Factorial Function into a Propositional L_A Function

as primitives (as emphasized by [Smith (2013)], section 5.3). Furthermore, we do not know the truth values of the arithmetical interpretation $\Im_A$ of L_A formulas in non-trivial cases. Thus, we cannot justify or control the translation procedure in such cases by independently comparing truth values.

The same argument applies regarding the second question. We might think to run a computation and compare the result of what the computation *does* with the truth value of what the corresponding L_A proposition *says* according to $\Im_A$. However, let us ignore any specific problems with this method since it cannot be effective in any case because of (iii): it is obvious that we cannot justify any general answer on the basis of specific values.

What we, in fact, do in justifying the correctness of the translation procedure is to relate *paraphrases* or *intended interpretations* of the relation between x and y in the recursively defined function to *paraphrases* or *intended interpretations* of the L_A expression $\phi(x, y)$. In doing so, one relates intended interpretations of the recursively defined functions in terms of *number-theoretic (or arithmetic) extensional functions* to the extensional understanding of the arithmetic interpretation $\Im_A$ of the corresponding propositional L_A functions.[7] In the 'Proof for (1)' quoted above, for example, Smith interprets $Sx = y$ as the successor function and $Z(x) = 0$ as the zero function: this is a number-theoretic interpretation since the values of the variables are numbers and not, e.g., symbols, and it is extensional since it does not matter how the values of the functions are assigned to their arguments. Such an interpretation is, indeed, best suited to be compared with the interpretation $\Im_A$ of the corresponding propositional functions. In the case of $Sx = y$ and `Sx = y`, one must judge that 'y is assigned to the successor to x' is true iff 'y is equal to the successor of x' is true. In the case of the zero function, one must compare '0 is assigned to x' and 'x is identical to itself and y equals 0'. The translation is correct iff the two paraphrases are coextensive. Note that the paraphrases depend on the syntax of the paraphrased expressions and differ to a greater extent as the expressions to be paraphrased become more complex. Consider, for example, the still rather 'easy' case of the translation of the factorial function into a Σ_1 formula, as presented in Table 1. The paraphrases of the expressions already differ significantly since the syntax of each paraphrased expression is completely different. The similarity of the paraphrases in the case of the successor function is an exception in this respect.

[7] Cf. the 'remarks about extensionality' by [Smith (2007)], pp. 29f., pp. 34f. and 35ff., and [Smith (2013)], chapter 14.4.

The assertion that the paraphrases of the initial functions are coextensive is, of course, rather trivial and beyond doubt when one takes the underlying semantics for granted. However, the implications of the method of evaluating the correctness of the translation are important: the 'criteria' for correctness are judgements concerning the coextensionality of intended paraphrases or interpretations. This method is not only the implicit method used in the proof of the initial functions; the same method is applied in the cases of composition, recursion and minimization, both by Smith and in alternative proofs of the *expressing theorem*. These proofs basically consist of defining *how* the recursive functions are translated; if the correctness of the translation procedure is addressed at all, then nothing more is done than to appeal to evidence based on one's semantic intuitions. Smith, for example, ends his translation of the factorial function into the formula given in Table 1 with the following emphatic statement: 'For this *evidently* expresses the factorial function' (my emphasis). Likewise, he ultimately 'proves' that the described procedure works in any arbitrary case by presenting a general translation schema φ and asserting that 'it is then *evident* that φ will serve to express the p.r. defined function f' ([Smith (2013)], p. 118, my emphasis). In the case of minimization, he again simply defines *how* minimization is translated into Σ_1 formulas and simply avows that this '*evidently* expresses' ([Smith (2007)], p. 277, my emphasis)[8] the function defined by minimization. Smith's proof is in no way less explicit than others, nor does it differ in method from other proofs of the *expressing theorem*. On the contrary, his proof is admirably clear and understandable.

My point of critique concerns not the complexity of the translation procedure but its application to hypothetical diagonal cases. I have no doubt that recursive functions such as the factorial function can be expressed and captured by the translation procedure. However, the question is whether it can be justified that such a procedure is *correct in any arbitrary case*, including the hypothetical case of a recursive definition of Q- or FOL-provability. Such a case inevitably results in a contradiction due to the diagonal context. Any generalization of the evidence gained from rather trivial, non-diagonal cases to arbitrary cases presumes a similarity between those cases. However, diagonal cases giving rise to contradictions are not similar to non-diagonal cases, which alone are addressed when appealing to semantic intuitions. This is why any sort of purported inductive proof concerns only the generality of the specified translation procedure and not its correctness.

According to the defined translation procedure, a presumed recursive property $Prov(x)$ can be related to a propositional function $P(x)$. The proof of the *expressing theorem* neither proves nor provides any evidence that $P(x)$ would indeed express $Prov(x)$ given that $Prov(x)$ exists. In this hypothetical case, the method of judging upon the equivalence of intended interpretations necessarily fails. By hypothesis, any intended interpretation of $Prov(\ulcorner\gamma\urcorner)$ must (i) be coextensive with 'the formula with number γ (i.e., $\neg P(\ulcorner\gamma\urcorner)$) is provable' and (ii) be expressed by $P(\ulcorner\gamma\urcorner)$. The interpretation $\Im_A$ of $P(\ulcorner\gamma\urcorner)$, however, cannot be coextensive with 'the formula with number γ (i.e., $\neg P(\ulcorner\gamma\urcorner)$) is provable' if Q is sound.

Of course, one can argue that this contradiction reduces the hypothetical assumption that Q-provability is recursively definable by $Prov(x)$ to absurdity. However, this argument is not conclusive because for it to be conclusive, it would be necessary to show that $Prov(x)$ would be expressible by $P(x)$ if $Prov(x)$ were to exist. In short, undecidability is shown only by independently proving expressibility; otherwise, one may likewise reduce to absurdity

[8]Cf. [Smith (2013)], p. 297: 'Intuitively, F expresses f; i.e. $f(m) = n$ iff $F(\overline{m}, \overline{n})$ is true (think about it!).'

the assumption that L_A is capable of expressing Q-provability or, more fundamentally, that FOL is capable of expressing FOL-provability.

Proofs of the *expressing theorem* do not explicitly address the question of how to prove the theorem for hypothetical diagonal cases. If this question is addressed, it becomes clear that any undecidability proof begs the question. Undecidability proofs, in fact, demonstrate that a diagonal case rules out the possibility of expressing and capturing the property in question. How can one justify that the *expressing theorem* holds for *all* recursively defined properties without presuming that any property that is demonstrably not expressible in L_A is not recursive? Circularity can be avoided only if it is conceded that the undecidability proof is underdetermined; under the assumption that Q is consistent, the diagonal case shows either that Q- and, consequently, FOL-provability cannot be defined recursively or that FOL-provability is not expressible within L_A and, consequently, FOL due to the diagonal case. Semantic evidence cannot settle the matter since our semantic intuitions are derived from normal cases that we can oversee and control.

Diagonal cases as considered in paradoxes such as the Liar Paradox, Russell's Paradox or Richards' Paradox show that our semantic intuitions are not reliable. Undecidability proofs resting on the technique for expressing diagonal cases in the language of logic do not prove that it is indeed possible in diagonal cases to express what one intends to say by means of the language of logic. The use of the diagonal method to prove theorems such as the unsolvability of the halting problem or Tarski's theorem and paradoxes such as the Liar Paradox or Richards' Paradox cast doubt on the use of the diagonal method to establish the undecidability of FOL. Interpreting Turing's and Church's proofs as undefinability proofs for FOL-provability within FOL is a straightforward alternative.

2.6. **Conclusion.** In addition to Turing's and Church's canonical proofs, many other versions of undecidability proofs of FOL (or fragments thereof) exist at present (cf. [Boerger et al. (2001)]). However, all of these undecidability proofs rest on the expression of undecidable problems within FOL and ultimately invoke diagonal cases. Any proof that it is possible to express undecidable problems within FOL is based on a claim of the general validity of translation procedures that encounter semantic problems in diagonal cases. Rigorous proofs concerning the possibility of defining a decision procedure for FOL should not be affected by such semantic problems. Instead of making a hopeless attempt to prove strong general claims concerning the expressibility of supposed functions within a language based on FOL, it would be more fruitful to attempt to solve the decision problem. Such an attempt concerns the possibility of algorithmizing the application of rules of a correct and complete calculus such that not only theoremhood but also non-theoremhood can be decided within a finite number of steps. This effort to seek a positive solution to the decision problem is not affected by problems arising from attempting to express the limits of logic within the language of logic. Whatever may result from this work, it can only increase our power to decide first-order logic formulas.

3. Wittgensteinian Background

The critique of Church's and Turing's proofs presented in the previous section is inspired by Wittgenstein. According to my understanding of Wittgenstein's remarks on the foundations of mathematics in general and undecidability proofs in particular, Wittgenstein

advocated a computational view of mathematics and logic and criticized undecidability proofs as underdetermined.

Unfortunately, Wittgenstein related his critique to Gödel's proof of the incompleteness of axiomatic systems of arithmetic (cf., in particular, RFM I, Appendix I and Part V, §§18f.) and not directly to Turing's or Church's proof. However, he did engage with the decidability of FOL in the first place (see section 3.2 below), and his critique concerns not the subtleties of the undecidability proofs but rather the principal question of whether such proofs are compelling. My critique of Church's and Turing's proofs is inspired by Wittgenstein's critique of Gödel's proof. However, in contrast to Wittgenstein, I claim that his kind of critique applies primarily to undecidability proofs of FOL. More precisely, it concerns the expressibility of a hypothetical decision function for FOL-provability by means of a propositional Σ_1 function in L_A, not Gödel's way of expressing unprovability by means of a Π_1 formula on the basis of expressing "y is a PA proof of x". Gödel's proof does not consider expressing provability as a computable function (cf. Def. 46 in [Gödel (1986)], p. 171, where Gödel defines 'x is a provable formula' in terms of a notion 'of which we cannot assert that it is recursive'). His proof is based on the much weaker assumption of recursively defining 'y is a proof of x' and expressing this recursive function as an open formula that he abbreviates as yBx. Provability (in PM or PA) is merely expressed by binding the free variable y in yBx. Gödel's proof derives a contradiction through diagonalizing $\exists yyBx$ and assuming it or its negation to be provable within an axiomatic system such as PM or PA. This contradiction can also be used to show that these axiomatic systems are incomplete (given their (ω-)consistency). Such an option is not available in the case of undecidability proofs of FOL since FOL can be conceptualized in an axiom-free form.

In relation to Gödel's proof, Wittgenstein stated that instead of reducing the assumption of decidability to absurdity, one could similarly "withdraw the interpretation" of G (cf. RFM I, Appendix I § 8 and § 10).[9] However, he did not distinguish between Gödel's Π_1 formula G and Σ_1 formulas that are intended to express recursive functions. Instead, his critique concerns the principal point that undecidability proofs leave open the possibility of rejecting the reliability of an intended interpretation in the diagonal case instead of reducing the assumption of decidability to absurdity. Wittgenstein emphasized that undecidability proofs are underdetermined, indirect proofs and that whether one is inclined to give up the search for a decision procedure due to such a proof depends on what one accepts as a criterion for proof (RFM I, Appendix I, §14f., and V, §22). Unfortunately, Wittgenstein argued very generally and did not relate his critique to specific assumptions of undecidability proofs that are, in fact, affected by his critique. The previous section intended to overcome this deficiency by relating Wittgenstein's critique to the *expressing theorem* in Church's proof and *Lemma 2* in Turing's proof. I consider this the most promising way to extract systematic value from Wittgenstein's critique.

However, much more important than the particular understanding of Wittgenstein's admittedly vague remarks on undecidability proofs is the fact that some of his key distinctions relating to mathematical and logical proofs suggest a serious alternative to the axiomatic method of mathematical logic. The following three sections sketch this alternative. The

[9]The exact understanding of this remark and the question of whether it is based on a profound understanding of Gödel's proof are extensively debated. I have discussed Wittgenstein's critique of Gödel's proof as well as the literature on this critique in detail in [Lampert (2018a)]. Unfortunately, I did not argue in that paper that Wittgenstein's critique should more correctly be related to undecidability proofs of FOL than to Gödel's proof.

specific critique of Church's and Turing's proofs in the last section and the motivation to explicate a decision procedure for FOL is best understood against this general background.

3.1. **Wittgenstein's Distinctions: Language of Computation vs. Language of Logic.** In his *Tractatus logico-philosophicus* from 1918, before the theory of computation was established, Wittgenstein drew several key distinctions, such as those between *operations and functions* (TLP 5.25; cf. TLP 5.2-5.3), between *formal and material concepts or properties/relations* (TLP 4.122-4.1274) and between *"showing" and "saying"* (TLP 4.121-4.1213, 4.126[3]). He accused Frege's and Russell's conception of mathematical logic (the "old logic", in Wittgenstein's words, as opposed to his conception of a "new logic") of not drawing these distinctions (TLP 4.122[3], 4.126[2], 4.1272[7], 5.25[3]; cf. PT 5.005341, CL letter 68 from 19.8.1919, p. 124).

According to Wittgenstein, basic arithmetic "functions", such as the successor, addition, multiplication, subtraction, and division functions, as well as basic so-called logical "truth functions", such as negation, conjunction, disjunction, implication, and quantification, are all *operations*. Operations can be (i) iteratively applied (whereas functions cannot, according to Wittgenstein's terminology; cf. TLP 5.251); (ii) reversed; (iii) the number of times they are applied can be counted; and (iv) their application can be limited in number or conditioned by the identification of some formal property. The application of operations starts from some input, and the applied operations compute an output. Arithmetic operations and logical operations form different internal related systems that manipulate different types of "forms": arithmetic operations generate numbers, whereas logical operations generate logical forms. Such forms are not referred to by symbols but rather are identified by their outer form and the syntactic rules applied to them.

Arithmetic properties (such as being a natural number), *arithmetic relations* (such as equality), *logical properties* (such as provability), and *logical relations* (such as logical equivalence) are all *formal* properties or relations. *Material* properties/relations are stated in the form of propositional functions, while *formal* properties/relations "are not expressed by means of a function" (TLP 4.126) but rather are identified by applying operations.

According to Wittgenstein, it cannot be "*said*" but rather is "*shown*" that formal properties or relations hold. One way to interpret this distinction is by claiming that formal properties are decidable by means of purely symbolic manipulation, while material properties are stated by interpreting propositional functions within a logical formalism. One cannot identify that material properties hold by means of purely syntactic operations that concern only the syntactic forms of symbols. Instead, material properties refer to objects; whether those objects have certain (material) properties is determined in accordance with the interpretation of symbols and can be proven only through axioms. By contrast, formal properties are not properties of objects that are referred to by symbols but rather are properties of the forms of symbols. The forms of symbols are identified by equivalence transformations within a formalism, e.g., by transformations of equalities in primitive arithmetic or by normal form transformations in logic. Formal properties are identified via a mechanical procedure that reduces members of equivalence classes to ideal representatives (TLP 6.113, 6.1203, 6.122, 6.126, 6.1265; cf. [Lampert (2017b)] and [Lampert (2018b)] for details).

Although Wittgenstein's conception of computation in terms of the application of operations shares certain basic intuitions with Church's or Turing's specification of computability, it resists the representation of computability within a logical formalism. Computability is

conceptualized in the form of purely syntactic manipulations of symbols, independent of any interpretation of those symbols and prior to any axiomatization. The distinction between computation and stating true or false propositions is a difference of the kind of language used, according to this analysis. This distinction of the language of computation from the language of logic means that the former language is autonomous and that any attempt to express formal properties such as provability with a language based on FOL and to capture those properties by means of axiomatic systems based on FOL is secondary.

The solvability of formal problems depends on the chosen language; as a prominent example, one might consider the language of algebra, which makes it possible to decide the possibility of geometric constructions.[10] In the case of undecidability proofs of axiomatic systems such as Q, there is no need to blame *the undecidability* of the underlying FOL (cf. [Smith (2013)], p. 302). Instead, it is the syntax and semantics of FOL that are to blame for the inability to express the decidability of FOL within FOL itself. Translating a decision procedure for FOL-provability into a propositional function that can be diagonalized via Gödelization results in intended interpretations that are not compatible with the syntax and semantics of FOL. From this point of view, FOL is the wrong language with which to adequately formalize a decision procedure for FOL. This argument questions not the usual syntax and semantics of FOL but rather the adequacy of their application to formalize a presumed decision procedure for FOL within a language based on FOL.

Computation as carried out by means of computer programs, Turing machines or recursive functions is based on the application of operations, in Wittgenstein's terminology. *Applying* operations is categorically different to *stating* or proving the truth of propositions. The syntax of computation is, roughly speaking, a syntax of recursive rules, not of propositional functions. While logical reasoning and, more specifically, automated theorem proving in FOL may well be computable, this activity is not fully expressible within the language of FOL, according to Wittgenstein's understanding of computation. If one does not assume but rather doubts that a decision procedure defined in the language of computation (be it ordinary code or code reduced to the language defined in terms of recursive functions or Turing machines) is adequately expressed in the syntax and semantics of propositional functions and captured in Q (or any consistent extension of Q), then one will most likely interpret undecidability proofs of FOL as reducing the assumption that languages based on FOL can be used to express a decision function for FOL, instead of the decidability of FOL, to absurdity.

Whereas tradition has it that the concept of a function is a general one, of which computable functions are a specification, Wittgenstein advocated for distinguishing the realm of the computable from the realm of propositions by syntactic means: Computable properties can be identified by defining operations, which are capable of iterated application (or "self"-application). However, such properties cannot be expressed by propositional functions since their intended interpretation in terms of self-reference is not reliable (cf. TLP 3.33-3.333). Given these distinctions, the endeavour to express and capture FOL-provability

[10][Kvasz (2008)] reconstructs the history of mathematics as a history of the development of the languages of mathematics that, step by step, made it possible to increase the power that can be achieved in expressing and solving problems. He offers numerous further examples that show the dependence of the expressibility and solvability of mathematical problems on the mathematical language used. Unfortunately, he does not consider the question of whether the Church-Turing theorem should alternatively be interpreted as showing the limits of the language of logic (FOL) with respect to expressing its own decidability, rather than stating that FOL is undecidable.

by means of a propositional function is based on a misconception; it is a kind of categorical mistake that is relevant as soon as diagonalization is of interest. This critique questions neither the informal concept of computation nor its explication according to Church's or Turing's thesis; rather, it questions the formal presentation of what is computable within the language of logic and the consequences thereof.

According to this critique, the decidability of FOL cannot be measured with respect to the possibility of expressing a decision procedure by means of a characteristic propositional function in some language based on FOL. As long as the decision problem is not solved positively, no hypothetical reasoning can settle the matter. From Wittgenstein's point of view, the decision problem is the problem of defining an algorithmic equivalence procedure within FOL that reduces FOL formulas to ideal symbols that enable one to decide formal properties such as provability or contradiction. Given the rationale for his critique, no meta-logical reasoning can rule out the possibility of a decision procedure that does not reach beyond FOL.

3.2. **Wittgenstein's Conjecture.** Based on his distinction between the realm of computation (what can be shown) and the realm of propositions (what can be said), Wittgenstein conjectured that FOL is decidable in a letter to Russell from 1913. Before he was confronted with the undecidability results of Gödel and Turing, he had already formed a principal and programmatic conception of computation and logic, although he was not interested in working out the technical details. In the late thirties, he was in close contact with Turing, and he was one of the first to whom Turing sent his paper from 1936/7. I have previously argued that he never abandoned his conjecture even after he was confronted with Turing's proof (cf. [Lampert (2019a)]).

However, Wittgenstein was neither interested in spelling out his programmatic conjecture nor interested in technically elaborating his basic distinctions, nor did he spell out his critique of undecidability proofs in detail. He seemed to simply be satisfied with his general conviction that the decidability of FOL cannot be measured by considering the (im)possibility of expressing the formal property of FOL-provability by means of a propositional function within a language based on FOL. This rather programmatic rejection of any negative solution to the decision problem is not satisfying. However, rejecting Wittgenstein's conjecture is not convincing, either, as long as this rejection simply takes the common acceptance of the Church-Turing theorem as its standard.[11] If one takes Wittgenstein's point of view seriously, then the Church-Turing theorem cannot be presumed when evaluating his conjecture. One should instead evaluate this challenge by evaluating attempts to demonstrate Wittgenstein's conjecture.

Discussions of foundational issues often end in spelling out various conceptualizations, aims, methods and standards that are difficult to evaluate without prejudice. Thus, prior to such a discussion, it is expedient to focus on the Church-Turing theorem as a decisive point of divergence on the shared background of the syntax and semantics of FOL. Since mathematical logic is a well-established discipline, a serious debate on its foundations must start from identifying some definite anomaly in any case.

This is the reason why I considered the first task in discussing Wittgenstein's rather programmatic view to be to attempt to elaborate a decision procedure for FOL. I started from

[11] Cf., among others, [Floyd (2005)], p. 95; [Potter (2009)], p. 181-183; and [Landini (2007)], p. 112-118.

the basic conviction that pure equivalence transformation within FOL is the sole foundation on which the algorithm should be based. This conviction is in line with Wittgenstein's conception of a logical proof. According to his conception of a new logic, a logical proof is not a derivation of theorems from axioms. Instead, more generally, a logical proof decides logical properties (such as provability or contradiction) by means of nothing other than the mechanical transformation of FOL formulas into ideal representatives of equivalence classes. These ideal representatives provide the criteria for identifying the logical properties of the formulas they represent. A positive solution to the decision problem reduces both provable (refutable) formulas and non-provable (non-refutable) formulas to expressions that allow one to identify the logical properties in question from the purely syntactic properties of the resulting expressions.

The FOL-Decider is the result of this work. Thus, this critique of the Church-Turing theorem can be either verified or falsified by checking this procedure and the proof of its logical foundations.

3.3. **A Modest Version of Wittgensteinian Finitism.** Rejecting the expressibility of a decision function for FOL within a language based on FOL (instead of rejecting the assumption of the decidability of FOL) is an essential feature of what I call "a modest version of a Wittgensteinian finitism". It is *modest* in the sense that it reduces Wittgenstein's rather general critique of mathematical logic and the axiomatic method to a specific critique of the Church-Turing theorem. It does not question the consistency of Q or consistent extensions of Q, nor does it question Gödel's incompleteness proof of PA. It also does not reject Church's or Turing's thesis, nor axiomatic or diagonal proof methods in general. The semantics and syntax of FOL, Q and PA are accepted, and meta-mathematical investigations are appreciated. All that is called for is a cautious application of the *expressing theorem* when diagonal cases are involved. Likewise, the diagonal method is unproblematic (or almost trivial) as long as no intended interpretations of diagonalized propositional functions are involved that are considered as the hypothetical results of a translation procedure for a decision procedure. Thus, neither Cantor's theorem nor the unsolvability of the halting problem is questioned, for example.

Instead, it is the Church-Turing theorem and its underlying assumption of the unrestricted expressibility of computable functions within the language of logic that are rejected. The elaboration of a decision procedure refers only to well-known and accepted proof methods, inference rules and notions of computation. Thus, this kind of critique is also *modest* in the sense that it applies only modest proof methods that make up the core of everyday mathematics and logic. It can be called *Wittgensteinian* since both the rejection of the general expressibility of computable functions within a language based on FOL and the conviction that FOL is decidable are motivated by ideas and distinctions presented by Wittgenstein.

The conception advocated for here can be called a version of *finitism* because it calls for a cautious use of generalizations. Generalization to an infinite number of cases presumes a certain similarity among the cases. However, this similarity is not a given when indifferently quantifying over "all interpretations", without distinguishing reliable and unreliable interpretations of FOL or L_A formulas. There is no basis for grounding the general claim of the *expressing theorem* on induction or some other sort of generalization from a finite number of well-defined cases when intended interpretations of diagonal cases are involved. The undecidability proofs of FOL and Q are based on strong, general claims that should

apply even to abnormal diagonal cases that are only hypothetically envisaged in indirect proofs. It is apparent that a proof is lacking for the strong claims stated in the *expressing theorem* or in *Turing's Lemma*, which hold only if these claims apply to the hypothetically assumed diagonal cases that induce contradictions.

Again, this critique is not based on a general rejection of any basic concepts or proof methods used in mathematics. It rejects neither the concept of (potential or even actual) infinity in general nor quantification over infinite domains in particular. Likewise, neither the diagonal method in general nor the diagonalization of propositional functions and the Diagonalization Lemma in particular is rejected here. Instead, all that is argued for is a cautious use of generalizations by showing how one can go astray in the case of the reasoning involved in undecidability proofs. However, being cautious in generalizing claims also means that one should not infer from this critique of the undecidability proof of FOL that the concepts and methods involved must be rejected from the outset. In this respect, the argument is, once more, *modest*.

Furthermore, a Wittgensteinian *finitism* is a version of *finitism* in the sense that it advocates for a computational conception of logic and mathematics *for its own sake* that cannot be threatened by limiting theorems proven by the axiomatic method. Solving mathematical and logical problems by means of finite computations is its ultimate aim since it "recognizes the uncertainty of all speculations" (Kronecker, cf. [Meschowski (1967)], p. 238, my translation).

In the following, I will characterize this version of finitism more generally as an alternative to axiomatic mathematics, an alternative for which I believe Wittgenstein advocated.

To more precisely specify this version of finitism, two forms of mathematics are to be distinguished by their proof methods and underlying languages: axiomatic mathematics and algorithmic (or computational) mathematics. Axiomatic mathematics was established, roughly speaking, by the developments in mathematics from 1850 to 1950. Frege and Russell established the language of logic as the universal formal language of all reasoning and, in particular, of mathematical reasoning. The invention of logical formalism went hand in hand with the development of set theory as a new ontology in which the language of logic is interpreted. This emerged as a new framework for formalizing informal mathematical reasoning. This framework gave rise both to new problems (such as questions of decidability and the limits of computation) and to new proof methods for solving those problems. Logical formalization, axiomatization and diagonalization were all established within this framework and used as new proof methods for investigating the limits of computation.

Algorithmic mathematics, in contrast, can be seen as a continuation of "old-fashioned", classical mathematics, seeking to solve mathematical problems by inventing new algorithms based on specific mathematical languages. It does not attempt to formally express and capture informal, ordinary reasoning through the use of a universal language; instead, it is oriented towards solving specific problems arising from the application of mathematics in science. While axiomatic mathematics interprets FOL as a universal language and aims for generality, algorithmic mathematics aims for computability by inventing specific algorithms designed to solve specific problems. Its language is the language of computation – program code, not FOL. The decision problem of FOL is merely one computational problem, in addition to others, that one should attempt to solve through computational logic by looking for specific logical transformations into ideal symbols in ideal logical notations. Algorithmic mathematics does not rely on meaningful propositions that state or describe some non-symbolic realm, nor does it deduce theorems from axioms; it instead solves formal problems,

identifies formal properties and relations, or constructs formal entities (such as numbers, sequences or geometric objects) through the pure manipulation of symbols. Wittgenstein's philosophy of mathematics can be read as a defence of the aims of classical, core mathematics in the face of the emergence of a new kind of higher mathematics that extends beyond what is computable.

While Wittgenstein's remarks ultimately express that he, at least to some extent, rejects the use of the language of logic in mathematics, he nevertheless pretends not to intend to revise mathematics. A modest version of Wittgensteinian finitism does not question the axiomatic method but does interpret its outcome as placing limits not on core, computable mathematics but rather on axiomatic mathematics pursued within the language of logic. The incompleteness of PA, first and foremost, says something about the limits of arithmetic when formulated in L_A (which is based on FOL) and axiomatized by PA; it does not say anything about attempts to pursue arithmetic by using other (non-logical) languages and other (non-logical) rules. According to an algorithmic point of view, FOL and arithmetic are separated by their languages. Quantification in a logical formalism is categorically distinguished from existence claims proven by constructive methods and from generality claims proven by induction in mathematics. Algebraic equations are interpreted not as propositional functions containing free variables that can be bound by logical quantifiers but rather as part of a specific algebraic language that calls for its own methods of computing solutions for the "unknowns". Arithmetic equality is a formal relation relating to numbers and distinguished from logical identity in terms of a propositional function referring to objects. Arithmetic and logical operations are not subsumed under a general concept of functions but rather (i) are separated as operations from an extensional, set-theoretic concept of functions and (ii) are separated from each other by constituting different sorts of languages.

The core of mathematics is computation, and this is an autonomous practice that cannot be limited by the axiomatic method. This method goes beyond the proof methods of computational mathematics and establishes its own limits. These limits are induced by the use of the language of logic, which is not suitable for expressing decision procedures or solving mathematical problems by computation in general. It is the intention to express formal properties by means of a logical syntax that necessitates to start from unproven axioms that can never fully capture what one intends to express within a language based on FOL.

A modest version of Wittgensteinian finitism (i) separates axiomatic mathematics from computational mathematics, (ii) interprets the limiting theorems of axiomatic mathematics as theorems concerning the framework of the axiomatic method, (iii) urges only cautious use of generalizations based on semantic intuitions and therefore questions the general validity of the *expressing theorem*, (iv) aims to approach mathematics and logic from a purely algorithmic point of view for its own sake, and (v) proves the merits of this approach by means of working out decision procedures based on modest proof methods, for example, a decision procedure for FOL on the basis of nothing but well-known rules of FOL.

Thus, a modest version of Wittgensteinian finitism does not question axiomatic mathematics as long as it does not pretend to place any limits on computational logic or mathematics. The Church-Turing theorem, however, does not merely state that the provability of FOL formulas cannot be expressed within a language based on FOL; instead, it claims that no computer program can be written to decide FOL. The FOL-Decider is designed to refute this claim, thereby demonstrating that finitism, in terms of computational logic and

mathematics, is able to achieve results that are impossible to achieve according to or within axiomatic mathematics.

Part 2. The FOL-Decider

4. The FOL-Decider as a Proof of Concept

The FOL-Decider is an implementation of the algorithm described in this paper, with the purpose of demonstrating the algorithm's feasibility. It is programmed in the Wolfram Language of *Mathematica*. This language permits a rather simple and transparent implementation. Only a slight effort has been made to optimize the decision procedure. This is because the intention is for the FOL-Decider to serve as a proof of concept (PoC) that focuses on the logical principles of the decision procedure rather than on its optimization. For this reason, the FOL-Decider is, in general, slower than well-known logic engines such as Beagle, Darwin, i-Prover, E, Vampire and SPASS. Short formulas with infinite models only, however, constitute a verifiable exception (cf. p. 66). The FOL-Decider accepts the TPTP syntax and runs under the TPTP site. It was tested with all FOL formulas in the TPTP library with fewer than 20 atoms, as well as other formulas. The restriction to formulas with fewer than 20 atoms is due to the disadvantageous properties of the FOL-Decider with regard to fast decision-making.

The general approach of the FOL-Decider is nothing special. Like all logic engines, it intends to refine the search for proofs within a correct and complete calculus to the effect that redundant and unnecessary application of deductive rules are avoided whenever possible. At best, satisfiability can be proven by showing that an exhausted (fully saturated) optimized proof search. The problem with any optimization of the proof search is to preserve correctness and completeness. Termination must not be achieved on the cost of these properties; this has to be proven in the case of any optimization of the proof search. Given the proof of the Church-Turing theorem is faulty, one cannot say in advance whether the endeavour to optimize proof search strategies has a principle limit. According to my point of view, saturation algorithms, which already prove satisfiability in many cases due to an optimized proof strategy, are on the way to a full decision procedure. The FOL-Decider can be conceived as a special version of a saturation algorithm, which provides sufficient principles to decide FOL based on a new proof strategy.

In contrast to powerful logic engines, the FOL-Decider does not make use of resolution and unification algorithms based on skolemization. I have attempted to translate the basic ideas of my decision procedure into linear resolution with subsumption. The proof search strategy used in the FOL-Decider is designed to detect superfluous repetitions of isomorphic proof steps in a proof search. The problem is to define precisely what constitutes a similar superfluous repetition of an isomorphic proof step in the case of a proof search within the resolution calculus for FOL. In a proof search based on resolution, (i) the lengths of resolvents and (ii) the depths of nested skolem functions may increase to infinity. There is no equivalent to (i) within the proof strategy of the FOL-Decider. Regarding (ii), I do not see how to translate the criterion for detecting repeated isomorphic proof steps (the *Loop Criterion*) that is used in the FOL-Decider into some similar, *general* and correct criterion that limits the depths of nested skolem functions.

Therefore, the FOL-Decider is based on a new proof strategy that I have developed for the purpose of specifying a decision procedure for FOL. I call this strategy "the ∧I-minimal

proof strategy in the NNF-calculus". It is based on nothing but equivalence transformation within FOL. This paper explains and proves this new strategy. Section 5 to section 7 introduce the basic relevant concepts and ideas. Section 8 explains the ∧I-minimal proof strategy. Section 9 defines the *Loop Criterion* and proves that the FOL-Decider terminates in every case. The final section, section 12, proves the correctness of the implemented decision procedure based on the explanations presented in the previous sections 10 and 11.

5. Preliminaries

The following informal description of the FOL-Decider explains the principles of the implemented decision procedure, and proves its termination and correctness. The proof depends not on algorithmic details but, rather, on the basic principles of the explained decision procedure. I will restrict the presented explanations to the critical second step of the FOL-Decider. The rather trivial first step, which concerns the conversion of FOL formulas into disjunctive normal forms of FOL (FOLDNFs), is explained in my papers [Lampert (2017a)] and [Lampert (2019b)]. In the following, I will invoke some of the essential definitions from these papers without repeating the trivial algorithms involved or proofs of the essential theorems. [Lampert (2019b)] introduces the proof strategy on which the FOL-Decider is based and illustrates it by applying it to Herbrand formulas, i.e., a fragment of FOL that is accepted to be decidable. Some familiarity with [Lampert (2019b)] will facilitate a better understanding of the paper at hand but is not necessary, as I summarize the basics in the following.

[Lampert (2017a)] provides some historical, philosophical and logical background concerning the use of FOLDNFs. The described algorithm for *minimizing* FOLDNFs[12], however, is irrelevant to the following. Only the algorithm for converting FOL formulas into FOLDNFs, described in section 2 of [Lampert (2017a)], is of importance for the FOL-Decider. In the following, I will presume the generation of FOLDNFs from initial FOL formulas and merely provide their definition, which is based on negation normal forms (NNFs) and primary formulas.

Definition 5.1. An FOL formula is expressed in *negation normal form* (NNF) iff it contains only ∧ and ∨ as dyadic connectives and ¬ appears only directly to the left of atomic propositional functions.

Definition 5.2. *Primary formulas* are defined as follows:

(1) An NNF that does not contain ∧ or ∨ is a primary formula.
(2) NNFs that contain ∧ or ∨ are primary formulas iff they satisfy the following conditions:
   (a) Any conjunction of n conjuncts ($n > 1$) is preceded by a sequence of existential quantifiers of minimal length 1, and all n conjuncts contain each variable of the existential quantifiers in that sequence.
   (b) Any disjunction of n disjuncts ($n > 1$) is preceded by a sequence of universal quantifiers of minimal length 1, and all n disjuncts contain each variable of the universal quantifiers in that sequence.
(3) Only NNFs that satisfy (1) or (2) are primary formulas.

[12]This algorithm is also implemented as a *Mathematica* program. Like the FOL-Decider, this program can be accessed and run via the link given in footnote 1.

Definition 5.3. An *FOLDNF* is a disjunction of length ≥ 1 of conjunctions of lengths ≥ 1 of primary formulas.

For FOLDNFs, I stipulate that all variables bound by universal quantifiers are x variables and that all variables bound by existential quantifiers are y variables. This can be easily achieved by renaming variables.

[Lampert (2019b)] describes essential proof strategies for the FOL-Decider and applies them to a fragment of FOL that is known to be decidable. This fragment consists of formulas that are convertible into FOLDNFs with primary formulas that do not contain $\vee$ in the scope of quantifiers. The decision procedure described in [Lampert (2019b)] is based on a procedure that decides, for two literals $L1$ and $L2$ of an FOLDNF, whether they are unifiable. A *unifiable pair of literals* is definable via *subformulas* ψ (cf. Definition 5.4). I will consider only unifiable pairs of literals of disjuncts D_i of an FOLDNF because an FOLDNF is refutable iff each disjunct D_i is refutable. Therefore, the decision problem for an FOLDNF reduces to the decision problem for D_i.

Definition 5.4. A *subformula* ψ is generated from a disjunct D_i of an FOLDNF as follows:

(1) Delete all literals in D_i except $L1$ and $L2$.
(2) Delete all quantifiers binding variables that do not occur in $L1$ or $L2$.
(3) Delete all occurrences of $\wedge$ and $\vee$ except the one that connects $L1$ and $L2$ in the logical hierarchy of D_i.
(4) If $L1$ and $L2$ are connected by $\vee$, replace this $\vee$ with $\wedge$.

Definition 5.5. A pair of literals $\{L1, L2\}$ from D_i is *unifiable* iff the subformula ψ generated from D_i that contains $L1$ and $L2$ is contradictory (cf. [Lampert (2019b)], section 12).

A pair of literals must be unifiable to contribute to a proof of contradiction.[13]

[Lampert (2019b)] describes a procedure for deciding whether a subformula ψ is refutable. The decidability of these formulas also trivially follows from the known decidability of Herbrand formulas. Thus, the unifiability of a pair of literals is decidable. Finally, [Lampert (2019b)] specifies an algorithm (Algorithm 13.2) that deletes from the disjuncts D_i of an FOLDNF all literals that are not members of any unifiable pair of literals. The resulting *purged* FOLDNF is sat-equivalent to the input formula ϕ. Furthermore, I assume that the FOLDNFs are *rectified*.

Definition 5.6. *Purged* and *rectified* FOLDNFs are FOLDNFs with disjuncts D_i that satisfy the following conditions:

Purged: Every literal in D_i is a member of at least one unifiable pair of literals from D_i.

Rectified: No variable in D_i is bound by more than one quantifier.

During the purging process, literals that are not members of any unifiable pair of literals from D_i are first replaced with *sat* (for "satisfiable"). Then, as many occurrences of *sat* as possible are deleted by applying the following two *sat*-rules (cf. Algorithm 13.2 from [Lampert (2019b)]):

[13] I speak of "proofs *of* contradiction" in the case of proofs that prove that an initial formula ϕ is contradictory (= refutable). By contrast, I refer to indirect proofs (= proofs by reductio ad absurdum) that prove the negation of one of the assumptions of a deduced contradiction as "proofs *by* contradiction". The FOL-Decider is concerned not with proofs *by* contradiction but with proofs *of* contradiction.

$$sat \wedge A \quad \dashv\vdash_{sat} \quad A \qquad \text{SAT1}$$
$$sat \vee A \quad \dashv\vdash_{sat} \quad sat \qquad \text{SAT2}$$

Table 2: *sat*-Rules

Step 1 of the FOL-Decider results in purged and rectified FOLDNFs that are sat-equivalent to the initial input formula ϕ. The universally quantified variables of these purged and rectified FOLDNFs are x variables with indices of depth 1, and the existentially quantified variables are y variables with indices of depth 1. Thus, if D_i contains m universal quantifiers and n existential quantifiers, then D_i contains the variables x_1 through x_m and y_1 through y_n. By applying several simple auxiliary rules (cf. table 4), the FOL-Decider also simplifies the FOLDNFs in this first step. However, the extensive algorithm for minimizing FOLDNFs described in [Lampert (2017a)] is not applied by default for reasons of efficiency.

Henceforth, for brevity, I will simply use the term "FOLDNFs" to refer to the purged, rectified and simplified FOLDNFs obtained as a result of step 1 of the FOL-Decider.

The refutability of certain initial formulas ϕ can already be decided in step 1 of the FOL-Decider by virtue of a simple False/sat-check. For any disjunct D_i of the resulting FOLDNF, the *DNF matrix* is generated, as defined below.

Definition 5.7. The *DNF matrix* of a formula ϕ is the scope of a prenex normal form of ϕ in disjunctive normal form (DNF).

If each disjunct of the DNF matrix contains two conjuncts, A and $\neg A$ (where A is atomic), then D_i is refutable. If D_i does not contain $\vee$ and it (and, consequently, the single disjunct of its DNF matrix) does contain a unifiable pair of literals, then it is refutable. In both cases, D_i can be deleted from the FOLDNF. If all of the D_i are refutable, then ϕ is refutable. If any disjunct of the DNF matrix of D_i does not contain any unifiable pair of literals from D_i, then ϕ is not refutable (i.e., ϕ is satisfiable). Based on this check, FOLDNFs with disjuncts D_i that do not contain $\vee$ are already decidable.

The algorithms applied in step 1 of the FOL-Decider are explained in the cited papers, where their correctness and termination are also proven. On this basis, the following (trivial) theorem can be proven:

Theorem 5.8. *The initial formula ϕ is sat-equivalent to its FOLDNF (= the result of step 1 of the FOL-Decider).*

Proof. The proof is based on the fact that all applied rules either (i) are logical equivalence rules, (ii) are sat-equivalent because only literals that are not part of a unifiable pair of literals are deleted, or (iii) rely on the described trivial False/sat-check. For details, cf. [Lampert (2019b)]. □

In the following, I consider only the second step of the FOL-Decider, which decides the refutability of disjuncts D_i whose refutability is not already decided in step 1. Such a disjunct (a conjunction of n primary formulas, where $n \geq 1$) contains at least one primary formula with at least one occurrence of $\vee$ that is preceded by at least one universal quantifier. As soon as it is decided that D_i is not refutable, the FOL-Decider returns `sat`; as soon as all D_i are identified as refutable, the FOL-Decider returns `False`.

6. Basic Idea

The type of decision-making performed for D_i in step 2 of the FOL-Decider is comparable to that of algorithms for deciding whether certain numbers are finitely representable within a specific notation. The division algorithm for deciding whether a rational number has a finite representation within the decimal system may serve as a simplest illustration. If a rational number is representable by a finite decimal, then the algorithm returns that finite decimal, e.g., 0.25 in case of $\frac{1}{4}$. If it is not, then the algorithm runs in an infinite loop consisting of infinite iterations of a computational step that yields an output that dictates that the same computation must be repeated; e.g., it repeats the computation $10 \div 3 = 3$ with remainder 1 in the case of $\frac{1}{3}$. Such a loop is detectable and can be utilized as a negative decision criterion; as soon as the computation enters such a loop, it can be concluded that no finite representation of that rational number is available in the decimal system. Consequently, the loop can be abandoned, and the computation terminates with a negative result as soon as the loop is detected.

In the case of the FOL-Decider, the question to be decided is not the equivalence of a number to a finite representation within a specific notation but rather the equivalence of a D_i to an *explicit contradiction*. This question is equivalent to the question of refutability.

Definition 6.1. An *explicit contradiction* is an NNF without universal quantifiers and with a DNF matrix in which each disjunct contains a conjunct A and a conjunct $\neg A$ (where A is atomic).

Thus, e.g., $\exists y_1 \exists y_2 (Fy_1y_2y_1 \wedge \neg Fy_1y_2y_1)$ is an explicit contradiction.

If an explicit contradiction is deducible from D_i, then the FOL-Decider returns a recipe for its deduction. If no explicit contradiction is deducible from D_i, then the steps of the proof computation along each proof path in the search tree for proofs of contradiction run, roughly speaking[14], in a detectable loop. The criterion for terminating the search for a proof on a given proof path due to the repetition of a step of the proof computation on that proof path is called the *Loop Criterion*. This criterion is defined in section 9, and it is proven to ensure the termination of the FOL-Decider in the case that no proof is found (cf. Theorem 9.18). If all proof paths for a D_i terminate without a proof of contradiction having been identified, then the FOL-Decider returns `sat`.

The general idea of proving that something is impossible by reducing endless iteration operations to "visual recursions" (detectable loops) was inspired by Wittgenstein's ideas on induction (cf. PR VIII-XIX, PG, part II.VI). The FOL-Decider applies this idea to identify unprovability by detecting loops in the proof search. The task is to define an algorithm in terms of an equivalence procedure for the application of logical operations of a correct and complete calculus such that the *Loop Criterion* can be applied to identify non-refutable formulas. As will be seen from the definition of the *Loop Criterion*, this approach still satisfies Wittgenstein's main idea of specifying a decision procedure by an algorithm such that the properties of the resulting expressions – *loop lists* in the case of the FOL-Decider – can serve as decision criteria. One might say that unlike in the axiomatic method, it is not properties that extend beyond logic that are encoded within logical formulas due to intended interpretations but rather *logical* properties that are encoded by expressions of the

[14]In fact, computations along proof paths that do not result in a proof are often terminated without application of the "Loop Criterion" (cf. Example 8.31, p. 55). However, the *Loop Criterion* ensures that every proof path will terminate unless a proof is found; cf. section 9.

proper *outer form* ("a feature of certain symbols", according to TLP 4.126) that result from a purely formal, intended equivalence transformation that does not reach beyond logic.

The rationale for the proof strategy underlying the FOL-Decider is to enable the correct application of the *Loop Criterion*. The proof strategy that makes this possible is the "∧I-minimal proof strategy". Before I explain the search for proofs based on this strategy in section 8, I will introduce the calculus that is applied in the proofs of the FOL-Decider in section 7.

7. NNF-Calculus

[Lampert (2019b)] introduces the NNF-calculus and proves its correctness and completeness (cf. Theorem 3.3. in [Lampert (2019b)]). The correctness follows trivially from the fact that all rules are well-known derivation rules. The completeness is proven by showing that any proof within the correct and complete tree calculus can be transformed into a proof in the NNF-calculus. This section summarizes the rules of the NNF-calculus. Its rules are applied either to NNFs or for the initial generation of NNFs. Connectives such as $\rightarrow$, $\leftrightarrow$, and $|$ are eliminated using their well-known definitions immediately at the beginning of step 1 of the FOL-Decider. In the following, I omit these rules as well as the *sat*-rules listed in table 2. In addition to the sat-equivalent elimination of literals during the purging process and the rule for universal quantifier elimination (cf. table 6), the NNF-calculus comprises only well-known logical equivalence rules. In the following, I assume that $\wedge$ ties more closely than $\vee$.

$\neg\neg A$	$\dashv\vdash$	A	DN
$\neg(A \vee B)$	$\dashv\vdash$	$\neg A \wedge \neg B$	DMG$\vee$
$\neg(A \wedge B)$	$\dashv\vdash$	$\neg A \vee \neg B$	DMG$\wedge$
$A \wedge B$	$\dashv\vdash$	$B \wedge A$	COM$\wedge$
$A \vee B$	$\dashv\vdash$	$B \vee A$	COM$\vee$
$(A \wedge B) \wedge C$	$\dashv\vdash$	$A \wedge (B \wedge C)$	ASS$\wedge$
$(A \vee B) \vee C$	$\dashv\vdash$	$A \vee (B \vee C)$	ASS$\vee$
$A \wedge (B \vee C)$	$\dashv\vdash$	$A \wedge B \vee A \wedge C$	DIS1
$A \wedge B \vee C$	$\dashv\vdash$	$(A \vee C) \wedge (B \vee C)$	DIS2
A	$\dashv\vdash$	$A \wedge A$	$\wedge$I

Table 3: Equivalence Rules from Propositional Logic

$A \wedge \neg A$	$\dashv\vdash$	$\bot$	IP$\bot$0
$A \vee \neg A$	$\dashv\vdash$	$\top$	IP$\top$0
$\top \wedge A$	$\dashv\vdash$	A	IP$\top$1
$\top \vee A$	$\dashv\vdash$	$\top$	IP$\top$2
$\bot \wedge A$	$\dashv\vdash$	$\bot$	IP$\bot$1
$\bot \vee A$	$\dashv\vdash$	A	IP$\bot$2

$$\begin{array}{rcll} (A \vee B) \wedge A & \dashv\vdash & A & \text{IP1}\wedge \\ A \wedge B \vee A & \dashv\vdash & A & \text{IP1}\vee \\ A \vee A & \dashv\vdash & A & \vee\text{I} \end{array}$$

Table 4: Auxiliary Rules

The auxiliary rules are applied in step 1 of the FOL-Decider to simplify the FOLDNFs. Like DN, DMG$\vee$, DMG$\wedge$, DIS1 and DIS2, these rules are not applied any further in step 2 of the FOL-Decider. ASS$\wedge$, ASS$\vee$, COM$\wedge$ and COM$\vee$ are applied implicitly in the FOL-Decider by defining conjunctions and disjunctions as "orderless". Of all the rules mentioned thus far, only $\wedge$I is used in step 2 of the FOL-Decider, as will be discussed below. In step 1 of the FOL-Decider, $\wedge$I is applied only in the unproblematic direction from right to left.

$$\begin{array}{rcll} \neg\exists\mu A(\mu) & \dashv\vdash & \forall\mu\neg A(\mu) & \text{Def. } \neg\exists \\ \neg\forall\mu A(\mu) & \dashv\vdash & \exists\mu\neg A(\mu) & \text{Def. } \neg\forall \\ \forall\mu\forall\nu A(\mu,\nu) & \dashv\vdash & \forall\nu\forall\mu A(\mu,\nu) & \forall V \\ \exists\mu\exists\nu A(\mu,\nu) & \dashv\vdash & \exists\nu\exists\mu A(\mu,\nu) & \exists V \\ \forall\nu(A \wedge B(\nu)) & \dashv\vdash & A \wedge \forall\nu B(\nu) & \text{PN1} \\ \forall\nu(B(\nu) \wedge A) & \dashv\vdash & \forall\nu B(\nu) \wedge A & \text{PN2} \\ \forall\nu(A \vee B(\nu)) & \dashv\vdash & A \vee \forall\nu B(\nu) & \text{PN3} \\ \forall\nu(B(\nu) \vee A) & \dashv\vdash & \forall\nu B(\nu) \vee A & \text{PN4} \\ \exists\nu(A \wedge B(\nu)) & \dashv\vdash & A \wedge \exists\nu B(\nu) & \text{PN5} \\ \exists\nu(B(\nu) \wedge A) & \dashv\vdash & \exists\nu B(\nu) \wedge A & \text{PN6} \\ \exists\nu(A \vee B(\nu)) & \dashv\vdash & A \vee \exists\nu B(\nu) & \text{PN7} \\ \exists\nu(B(\nu) \vee A) & \dashv\vdash & \exists\nu B(\nu) \vee A & \text{PN8} \\ \forall\nu(A(\nu) \wedge B(\nu)) & \dashv\vdash & \forall\nu A(\nu) \wedge \forall\nu B(\nu) & \text{PN9} \\ \exists\nu(A(\nu) \vee B(\nu)) & \dashv\vdash & \exists\nu A(\nu) \vee \exists\nu B(\nu) & \text{PN10} \\ \exists\mu A(\mu) & \dashv\vdash & \exists\nu A(\nu) & \text{SUB1} \\ \forall\mu A(\mu) & \dashv\vdash & \forall\nu A(\nu) & \text{SUB2}^{15} \end{array}$$

Table 5: Equivalence Rules for Predicate Logic

Like DN, DMG$\vee$ and DMG$\wedge$, the quantifier definitions Def. $\neg\exists$ and Def. $\neg\forall$ are applied only to obtain NNFs at the beginning of step 1 of the FOL-Decider. Similar to the associative and commutative laws for $\wedge$ and $\vee$, $\forall V$ and $\exists V$ are applied implicitly in the FOL-Decider by defining sequences of similar quantifiers as orderless. The application of PN laws, however, is significant in both step 1 and step 2 of the FOL-Decider. By applying PN laws from left to right, anti-prenex normal forms can be generated. In anti-prenex normal forms, quantifiers are driven inward as far as possible through the application of PN laws. This, in turn, makes it possible to generate optimized prenex normal forms by applying PN

[15]The following restriction holds for SUB1 and SUB2: ν does not occur in $A(\mu)$.

laws from right to left. In optimized prenex normal forms, existential quantifiers are placed as far to the left of universal quantifiers as possible by means of PN laws (cf. Definition 8.8). This will be discussed below on p. 42 (cf. also [Lampert (2019b)], section 6).

SUB1 and SUB2 are applied during the rectification process in step 1 of the FOL-Decider. To optimize the application of the auxiliary rules, SUB1 and SUB2 are also applied in step 1 of the FOL-Decider to achieve the opposite of rectification, namely, to use as few different variables as possible. This serves merely to simplify the FOLDNFs and is not essential for the decision procedure. In step 2 of the FOL-Decider, SUB1 and SUB2 are used only for rectification subsequent to the application of ∧I (from left to right). Unlike in step 1, in step 2, the depth of the indices is incremented by 1 with each application of SUB1 or SUB2. For example, after ∧I has been applied once to multiply a universally quantified expression $\forall\mu A(\mu)$, the variable μ is replaced with μ_1 in the first conjunct and with μ_2 in the second. Renaming by increasing the depth of the variables makes it possible to identify the relations of the variables, literals and pairs of literals that arise from the multiplication of conjuncts via ∧I with the initial variables, literals and pairs of literals from the initial D_i in step 2 of the FOL-Decider. Indices of depths > 1 indicate *derivates*.

Definition 7.1. A *derivate* of a variable μ with an index of depth 1 is a variable that is identical to μ up to indices of depth 1. A *derivate* of a pair of literals $\{L1, L2\}$ containing variables with indices of depth 1 is a pair of literals that is identical to $\{L1, L2\}$ up to indices of depth 1.

For example, x_{1_2} is a derivate of x_1, and $\{Fx_{1_2}y_{2_1}, \neg Fx_{3_{2_1}}x_{2_{1_1}}\}$ is a derivate of $\{Fx_1y_2, \neg Fx_3x_2\}$.

Aside from the *sat*-rules (cf. table 2), the following rule is the only rule in the NNF-calculus that is not a logical equivalence rule:

$$\exists\mu \ldots \forall\nu A(\mu, \nu) \quad \vdash \quad \exists\mu A(\mu, \nu/\mu) \quad \forall\text{E}$$

Table 6: Universal Quantifier Elimination

For simplicity, it is not required that $\forall\nu$ must occur directly to the right of $\exists\mu$. It is required only that $\forall\nu$ be in the scope of $\exists\mu$. Upon the application of ∀E, all occurrences of ν in the scope of $\forall\nu$ are replaced with μ, and $\forall\nu$ is eliminated. It is also permissible to replace ν with μ when $\exists\mu$ is a *new* existential quantifier preceding the resulting formula. I subsume this case under ∀E. I arbitrarily choose the variable y_0 as a new y variable and require that y_0 does not occur in the expression to the left of $\vdash$ in ∀E.

Step 2 of the FOL-Decider starts from the D_i; then, ∧I is iteratively applied to universally quantified expressions. Each application of ∧I is followed by miniscoping (by applying laws PN1-8 from left to right) and rectification (by applying SUB1 and SUB2). I denote anti-prenex expressions that result from ∧I applications, miniscoping and rectification by D_i^*. In contrast to step 1 of the FOL-Decider, in step 2, neither DIS1 or DIS2 nor PN9 or PN10 is applied to generate anti-prenex normal forms D_i^* (cf. also p. 43 below). Strictly speaking, the application of ∧I results in a conjunction of identical conjuncts. Henceforth, however, I subsume miniscoping and rectification under the term "∧I application". Thus, the resulting conjuncts are not identical because they vary in the indices of the variables that are bound by quantifiers occurring in the multiplied expression. Since only universally quantified expressions are multiplied in step 2 of the FOL-Decider, the resulting conjuncts

vary by at least one universally quantified variable. In the case that a proof is found, the final D_i^* is also "purged*", meaning that literals that are not needed for the ∧I-minimal proof are eliminated from D_i^* (cf. Definition 8.6). As long as no final D_i^* has been derived, the D_i^* are purged* only for the sake of generating optimized prenexes from *purged** anti-prenex normal forms in the steps of the proof calculation in step 2 of the FOL-Decider; cf. footnote 26 for an example. I denote optimized prenex normal forms generated from purged* D_i^* by D_i^{**}. An optimized prenex normal form D_i^{**} is optimal iff it allows an explicit contradiction to be deduced by applying ∀E (cf. Definition 8.22 for details). Finally, I use D_i^{***} to denote explicit contradictions deduced from D_i^{**} by applying ∀E (cf. figure 1).

Step 2 of the FOL-Decider computes whether explicit contradictions D_i^{***} can be deduced from the initial D_i by applying ∀E to optimized prenex normal forms D_i^{**}; this computation is performed by applying ∧I and generating D_i^*. Strictly speaking, ∀E is never actually applied in the FOL-Decider. The FOL-Decider merely computes whether applications of ∀E subsequent to equivalence transformations *would* result in a proof of contradiction. Thus, ∀E is never applied to formulas that are not refutable. Moreover, in ∧I-minimal proofs consistent with the recipe returned by the FOL-Decider, ∀E will only be applied to formulas identified as refutable if doing so will ultimately allow explicit contradictions to be deduced. Thus, the decision-making of the FOL-Decider is based on nothing but equivalence transformations with regard to the property of refutability, which is the property in question.

There is no need for a rule for existential quantifier elimination in the NNF-calculus. Unlike in the tree calculus (tableau calculus), quantified expressions are not decomposed multiple times. Instead, in the proofs of the NNF-calculus, universally quantified expressions $\forall\mu A(\mu)$ are multiplied by applying ∧I. Consequently, the universal quantifier $\forall\mu$ and its variable μ are multiplied. After rectification and the subsequent suitable generation of an optimized prenex normal form, the multiplied x variable can then be replaced with different y variables in literals stemming from different conjuncts by applying ∀E (cf. Example 4.1, [Lampert (2019b)], p. 7, and the discussion on p. 49 below). I call the process of computing and applying ∧I in step 2 of the FOL-Decider "∧I-optimization".

In the framework of the NNF-calculus, the decision problem for D_i consists of defining a criterion for the ∧I applications performed to achieve multiplications of universally quantified expressions that are necessary and sufficient to find a proof of contradiction via *unification* by applying ∀E. In contrast to Definition 2.3 of [Lampert (2019b)], which defines unification for *single* pairs of literals of a subformula ψ, the following Definition 7.2 defines unification for a *set* of unifiable pairs of literals.[16]

Definition 7.2. *Unification* is the result of replacing (universally quantified) x variables such that identical positions[17] in a unifiable pair of literals from D_i^* are occupied by identical (existentially quantified) y variables. A *set* of unifiable pairs of literals from D_i^* is *unifiable*

[16]In contrast to Definition 2.3 of [Lampert (2019b)], which implies that *unification* is a logically valid process in the case of a *single* pair of literals, Definition 7.2 does not presuppose that the unification of *all* pairs of literals is logically valid. It merely presupposes, in accordance with Definition 2.3 of [Lampert (2019b)], that the unification of each *single* pair of literals from a set of unifiable pairs of literals in the corresponding subformula ψ is logically valid. In contrast to the *unification* of a set of pairs of literals, the *unifiability* of such a set presupposes a logically valid procedure according to Definition 7.2. Proofs of contradiction depend on the unifiability of sets of pairs of literals.

[17]I identify positions in pairs of literals with respect to the locations of the arguments of the corresponding predicates of the literals. Thus, x_1 and y_1 occur in identical positions in $\{Fx_1, \neg Fy_1\}$.

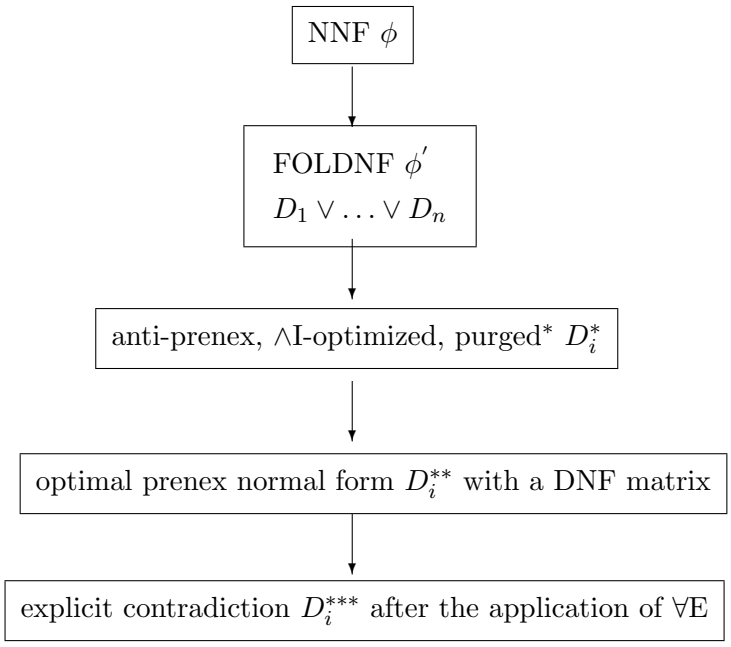

Figure 1: Steps of a proof in the NNF-calculus

if identical y variables can be made to occupy identical positions in all pairs of literals in that set by applying a logically valid substitution procedure. A *set* of unifiable pairs of literals is *unified* if all identical positions in all pairs of literals are occupied by identical y variables.

In proofs within the NNF-calculus, identical positions in literals are unified by applying ∀E. Step 2 of the FOL-Decider computes, prior to any application of ∀E, whether (minimal) sets of pairs of literals that would suffice to enable the deduction of explicit contradictions D_i^{***} are *unifiable* within the (correct and complete) NNF-calculus through the generation of D_i^* and D_i^{**} starting from D_i.

8. Searching for ∧I-minimal Proofs

8.1. **∧I-minimal Proofs.** In step 2 of the FOL-Decider, the only rule applied that increases the complexity is ∧I in the direction from left to right. ∧I is applied to replace a universally quantified x variable in different conjuncts with different existentially quantified y variables for the purpose of unification. In step 2 of the FOL-Decider, different proof paths are generated to enable a systematic search for ∧I-minimal proofs. Each proof path is determined by a specific combination of ∧I applications to unify pairs of literals for the deduction of explicit contradictions via the application of ∀E. According to the completeness theorem of the NNF-calculus (Theorem 3.3 in [Lampert (2019b)]), a proof based on ∧I applications in the NNF-calculus exists if D_i is refutable. The search for proofs in the NNF-calculus can be reduced to a search for ∧I-minimal proofs.

Definition 8.1. A proof of contradiction in the NNF-calculus is *∧I-minimal* iff each application of ∧I is necessary. A ∧I application in a proof of contradiction for D_i is *necessary* iff each conjunct in the final D_i^* resulting from that ∧I application is necessary. A *conjunct*

in the final D_i^* is *necessary* iff eliminating it causes no explicit contradiction D_i^{***} to be deduced any longer.

One can decide whether the condition for a necessary conjunct in the final D_i^* is satisfied as follows. Let C be a conjunct in the final D_i^* that results from a $\wedge$I application. Let `lits` denote the literals in D_i^{***} that are generated from the literals in C by replacing x variables with y variables. Let M be the DNF matrix from D_i^{***}, and let M' be the matrix that is obtained from M if one eliminates the literals `lits` in M. Then, C in D_i^* is necessary for the proof of contradiction iff it is not the case that each disjunct of M' contains two conjuncts, A and $\neg A$; cf. Example 5.2 in [Lampert (2019b)].

Theorem 8.2. *If D_i is refutable, then a $\wedge$I-minimal proof of D_i exists.*

Proof. From the completeness of the NNF-calculus (cf. Theorem 3.3 in [Lampert (2017a)]), it follows that a proof in the NNF-calculus exists if D_i is refutable. Any proof of D_i in the NNF-calculus, however, can be reduced to a $\wedge$I-minimal proof by eliminating unnecessary $\wedge$I applications. Therefore, a $\wedge$I-minimal proof of D_i exists if D_i is refutable. □

The FOL-Decider calculates only necessary and sufficient conditions for $\wedge$I-minimal proofs. Either it terminates the search for a proof on a particular proof path, if a necessary condition for a $\wedge$I-minimal proof is not satisfied, or it returns a recipe for either (i) a $\wedge$I-minimal proof (normal case) or (ii) an overdetermined $\wedge$I-minimal proof that contains a $\wedge$I-minimal proof (abnormal case). The latter may arise due to insufficient optimization of the $\wedge$I-minimal proof strategy such that the proof search is not reduced to $\wedge$I-minimal proofs only. For the question of decidability, it is sufficient to reduce the search space to a finite one that contains all $\wedge$I-minimal proofs and to identify either a $\wedge$I-minimal proof or an overdetermined $\wedge$I-minimal proof within this finite search space (cf. section 11).

Step 2 of the FOL-Decider generates anti-prenex disjuncts D_i^* with universally quantified expressions that have been multiplied due to $\wedge$I applications. For simplicity, I also subsume D_i under expressions of the D_i^* type. From this perspective, D_i is the result of applying $\wedge$I 0 times. The FOL-Decider does not, in fact, generate D_i^{**} or D_i^{***}. In the case that a proof of contradiction for D_i is found, it returns (i) the final purged*[18] anti-prenex normal form D_i^*, (ii) the final minimal set `L` of pairs of literals from D_i^* that constitute the identified $\wedge$I-minimal proof of contradiction, (iii) a final substitution list σ that specifies how the x variables should be replaced with y variables in `L` to unify all literals from `L`, and (iv) the final prenex $\wp$ of D_i^{**} that allows $\forall$E to be applied to replace x variables with y variables as prescribed by σ and, consequently, to generate D_i^{***}.

8.2. **Parameters of a $\wedge$I-minimal Proof Search: `L`, $\wp$, and `sub`.** Step 2 of the FOL-Decider calculates the $\wedge$I applications that are necessary in the search for a $\wedge$I-minimal proof. Each proof step on a proof path consists of an application of $\wedge$I. Its calculation depends on the following parameters:

(1) a minimal set `L` of unifiable pairs of literals,
(2) an optimized prenex $\wp$,
(3) a specification `sub` of substitutions that unify the single pairs of literals in `L`, and
(4) a loop list.

[18] Cf. p. 38 and Definition 8.6.

From the substitution specification `sub`, a substitution list σ can be generated that specifies, for each x variable ν, the y variables with which ν must be replaced to unify all pairs of literals in `L`.

Remark 8.3. If `sub` specifies that an x variable `xvar` must be replaced with several y variables `yvars`, this means that instances of `xvar` in different literals must be replaced with different y variables from `yvars` to unify all pairs of literals in `L`. In accordance with the ∧I-minimal proof strategy, this unification can be achieved in a logically valid way only through ∧I applications.

The first three parameters specify the substitutions of the x variables and, consequently, the applications of ∧I. Alternative specifications of these parameters result in alternative proof paths. This section explains these three parameters in detail, whereas section 9 discusses the loop list.

Definition 8.4. A set `L` of unifiable pairs of literals from D_i^* is *minimal* iff (i) unification of these pairs of literals from D_i^* is sufficient to satisfy the condition that each disjunct of the DNF matrix of D_i^* contains two contradictory literals, A and $\neg A$, and (ii) said condition is no longer satisfied if any one of these pairs of literals is eliminated from `L`.

Thus, the unification of `L` is *minimally sufficient* to satisfy the condition that each disjunct of the DNF matrix contains an explicit contradiction.

Definition 8.4 does not presuppose that it is possible to unify all pairs of literals from `L` in a logically valid way by deducing an explicit contradiction D_i^{***} from D_i^*. It presupposes only that each *single* pair of literals can be unified in a logically valid way by deducing an explicit contradiction from the corresponding subformula ψ. As will be explained in more detail later in this section, the parameters `L`, $\wp$ and `sub` are defined such that one can conclude the (logically valid – cf. Definition 7.2) unifiability of *all* pairs of literals from `L` *as a whole* only if no x variable in `L` must be replaced with more than one y variable in order to unify all pairs of literals in `L` according to `sub`. If the unification of `L` requires any x variable to be replaced with more than one y variable according to `sub`, then ∧I must be applied. Thus, within the framework of the ∧I-minimal proof strategy, whether all pairs of literals of a minimal set `L` are unifiable can be decided only by applying ∧I. This ∧I application may cause it to be necessary to extend `L` in order to satisfy condition (i) specified in Definition 8.4.

Remark 8.5. The unification of each pair of literals in a minimal set `L` of unifiable pairs of literals is a *necessary* condition for a ∧I-minimal proof on a given proof path. In the case that no x variable must be replaced with more than one y variable according to `sub`, I call `sub` "unambiguous". As will be shown below on p. 48, the unambiguity of `sub` is a *sufficient* condition for the identification of a ∧I-minimal proof.

By referring to `L`, one can define purged* anti-prenex normal forms as follows:

Definition 8.6. An anti-prenex normal form D_i^* is *purged** iff D_i^* contains only literals from `L`.

The *Fundamental Principle* of the ∧I-minimal proof strategy underlying Definition 8.1 can be specified with reference to `L`. To do so, I first define "selected conjuncts". In this definition, I presume that conjunctions of length m ($m \geq 1$) contain m conjuncts.

Definition 8.7. *Selected conjuncts* are (i) conjuncts from D_i^* that contain literals in L and (ii) all conjuncts resulting from a ∧I application.

This definition relates to any stage of the proof. It applies to any modification of D_i^* and L.

The *Fundamental Principle* can be defined in relation to selected conjuncts (Definition 8.7) and necessary conjuncts (Definition 8.1) as follows:

Fundamental Principle (FP). *Any once-selected conjunct must be necessary for a ∧I-minimal proof.*

Starting from the conjuncts in D_i prior to any application of ∧I, FP necessitates that once they have been selected, conjuncts will never be "dropped" in the proof search. In the course of further ∧I applications on a proof path, the selected conjuncts can only be further multiplied. Every step on a proof path involves the replacement of one selected conjunct with n ($n > 1$) conjuncts. Consequently, L must always contain at least one literal from each conjunct that results from an application of ∧I. If this is not the case, the proof path can be terminated because it will not result in a ∧I-minimal proof (cf. Example 8.28, p. 53 below). If a proof is found, it is based on all selected conjuncts and, therefore, all ∧I applications along the proof path.

To find a ∧I-minimal proof if it exists, the FOL-Decider generates *all* minimal sets L. Different sets L generated from D_i^* correspond to different alternative proof paths. ∧I-minimal proofs are restricted to *minimal* sets L to keep the required number of substitutions low. Consequently, the number of ∧I applications needed to replace x variables with different y variables is kept low.

Optimized prenexes $\wp$ are prenexes with universal quantifiers as far to the right of existential quantifiers as possible.

Definition 8.8. A prenex $\wp$ of a prenex normal form D_i^{**} that is generated from the anti-prenex normal form D_i^* is *optimized* iff the universal quantifiers are in the scope of the existential quantifiers to the greatest possible extent through the application of PN1-8.

Algorithm 10.1 in [Lampert (2019b)] defines the process of generating all possible optimized prenex normal forms from a subformula ψ. In the case of anti-prenex normal forms D_i^* that also contain ∨, this algorithm must be extended such that PN3-4 and PN7-8 are also considered. Because this extension is trivial and adds nothing fundamentally new, I abstain here from presenting the full algorithm for generating all optimized prenexes from a given anti-prenex normal form D_i^*. In fact, the FOL-Decider applies a procedure that is more effective than Algorithm 10.1 in [Lampert (2019b)]. In the FOL-Decider, prenexes $\wp$ are represented as lists of x and y variables. Optimized prenexes are generated combinatorially from different sequences of x and y variables. In doing so, the algorithm refers to *purged** anti-prenex normal forms D_i^* that contain only literals from L. The key to the ∧I-minimal proof strategy is that generating prenex normal forms from *anti-prenex* normal forms makes it possible to generate all and only *optimized* prenexes with universal quantifiers that are in the scope of the existential quantifiers to the maximal extent. This increases the possibility to apply ∀E for unification without needing to apply ∧I (cf. section 6 in [Lampert (2019b)] and p. 48 below). Alternative possibilities for generating optimized prenexes from anti-prenex normal forms give rise to alternative proof paths.

Step 1 of the FOL-Decider increases the number of quantifiers through the application of PN9 and PN10 and through the conversion of the scope of universal quantifiers into

conjunctive normal form (CNF) and the scope of existential quantifiers into DNF. This is done so that the scope of quantifiers is minimized to the maximal extent when the PN laws are applied. For simplicity, these strategies are not applied in step 2 of the FOL-Decider when generating anti-prenex normal forms.[19] Step 2 of the FOL-Decider increases the number of quantifiers and the number of variables they bind only as a result of $\wedge$I applications. Therefore, the strategy for minimizing the number of necessary $\wedge$I applications in a proof of contradiction is not absolute but instead depends on D_i^* and D_i^{**}, in which the number of quantifiers does not change during the process of miniscoping or prenexing.[20]

Henceforth, I use `L` to refer to *minimal* sets of pairs of literals and $\wp$ to refer to *optimized* prenexes.

`sub` specifies substitutions to be made in order to unify pairs of literals in `L`. Two cases must be distinguished when specifying these substitutions. In *case 1*, a single pair of literals unambiguously determines a specific y variable that must replace an x variable in a certain position to unify that pair of literals. In *case 2*, a single pair of literals merely determines that two x variables, occurring in identical positions in $L1$ and $L2$, must be replaced with *the same* y variable in those positions, without dictating any specific y variable for that purpose. *Case 1* is illustrated by $\{Fx_1, \neg Fy_1\}$; *case 2*, by $\{Fx_1, \neg Fx_2\}$. In *case 2*, the substitutions of the x variables may be determined by substitutions of x variables in *other* unifiable pairs of literals in `L` (*case 2.1*), or they may not (*case 2.2*). In *case 2.1*, multiple alternative possibilities for replacing x variables with y variables for unification may arise.

Example 8.9. Let D_i be given as follows:

$$\exists y_1 G y_1 y_1 \wedge \forall x_1 (F x_1 \vee \forall x_3 (\forall x_4 H x_1 x_3 x_4 \vee J x_1 x_3)) \wedge$$
$$\forall x_2 \exists y_2 (\neg F y_2 \wedge \forall x_5 (\neg G x_2 x_5 \vee \neg H x_2 y_2 x_5) \wedge \neg J x_2 y_2) \tag{8.1}$$

There is exactly one minimal set `L` of unifiable pairs of literals for (8.1):

$$\{\{Fx_1, \neg Fy_2\}, \tag{8.2}$$
$$\{Gy_1y_1, \neg Gx_2x_5\}, \tag{8.3}$$
$$\{Hx_1x_3x_4, \neg Hx_2y_2x_5\}, \tag{8.4}$$
$$\{Jx_1x_3, \neg Jx_2y_2\}\} \tag{8.5}$$

There is also only one optimized prenex $\wp$ for (8.1), which is represented by a list of x and y variables, according to the FOL-Decider:

[19]The number of $\wedge$I applications can also be decreased by minimizing the scope of existential quantifiers above conjunctions to the extent that this is sat-equivalent (cf. the discussion of $\exists$M optimization in section 7 of [Lampert (2019b)]). For simplicity, however, the FOL-Decider makes use of $\exists$M optimization only to identify unifiable pairs of literals. $\exists$M optimization is applied neither when generating FOLDNFs in step 1 nor when generating anti-prenex normal forms D_i^* in step 2. Nevertheless, $\exists$M optimization is an *optional* tool in step 1 of the FOL-Decider (cf. also footnote 9 in [Lampert (2019b)]).

[20]The number of $\wedge$I applications on a proof path is also related to the extent to which the FOLDNFs and D_i^* are simplified. The number of $\wedge$I applications could also, of course, be minimized via extensive minimization procedures such as those described in [Lampert (2017a)]. However, only rare use is made of the simplification of FOLDNFs in step 1 of the FOL-Decider. In step 2, the D_i^* are merely simplified by deleting literals that are not included in `L` (purging) to generate optimized prenexes $\wp$ or to return the final D_i^*.

$$\{y_1, x_2, y_2, x_1, x_3, x_4, x_5\} \tag{8.6}$$

In (8.2), x_1 must be replaced with y_2; in (8.3), x_2 must be replaced with y_1, and x_5 must be replaced with y_1; in (8.4), x_1 and x_2 must both be replaced with the same y variable, x_3 must be replaced with y_2, and x_4 and x_5 must both be replaced with the same y variable; and in (8.5), x_1 and x_2 must both be replaced with the same y variable, and x_3 must be replaced with y_2. In contrast to x_3, x_4 and x_5, at least one of the x variables x_1 and x_2 must replaced with more than one y variable to unify all pairs of literals in (8.2) to (8.5). Their possible substitutions in identical positions in (8.4) and (8.5) are determined by the specifications of their substitutions in (8.2) and (8.3). Since x_1 must be replaced with y_2 in (8.2) and x_2 must be replaced with y_1 in (8.3) (*case 1*), four possible substitutions for x_1 and x_2 in (8.4) and (8.5) arise through combinatorial means (*case 2.1*):

(1) x_1 and x_2 may be replaced with y_1 in both (8.4) and (8.5),
(2) x_1 and x_2 may be replaced with y_2 in both (8.4) and (8.5),
(3) x_1 and x_2 may be replaced with y_1 in (8.4) and with y_2 in (8.5), or
(4) x_1 and x_2 may be replaced with y_2 in (8.4) and with y_1 in (8.5).

(1) necessitates the replacement of x_1 with y_1 in addition to its replacement with y_2 in (8.2) and (2) necessitates the replacement of x_2 with y_2 in addition to its replacement with y_1 in (8.3). In both cases, however, only x_1 (in (1)) or only x_2 (in (2)) must be replaced with more than one y variable to unify the pairs of literals in `L`. By contrast, (3) and (4) necessitate the replacement of both x_1 and x_2 with both y_1 and y_2. The FOL-Decider realizes *all* of the combinatorially possible substitutions for *case 2.1* along different proof paths. Thus, specifications `sub` such as (3) and (4) that result in more substitutions than other possible specifications `sub` such as (1) and (2) are not excluded. The reason for this is that it cannot be guaranteed that an initial specification `sub` that requires more applications of ∧I may not result in a `sub` that requires fewer applications of ∧I at a later stage along the proof path.[21] If all combinatorially possible substitutions for *case 2.1* are generated in alternative specifications `sub`, then no ∧I-minimal proof will be excluded. It might be possible to define

[21]This is due to minimal extensions, as necessitated by ∧I applications, of minimal sets `L` of literals. The following example illustrates such a case.

Example 8.10. Let D_i be given as follows:

$$\forall x_1 \exists y_1 F x_1 y_1 \wedge \forall x_2 \forall x_5 (\forall x_7 (F x_5 x_7 \vee \neg F x_2 x_7) \vee \neg F x_5 x_2) \wedge$$
$$\forall x_3 (\exists y_2 (F x_3 y_2 \wedge G y_2) \vee G x_3) \wedge \forall x_4 (\forall x_6 (\neg F x_4 x_6 \vee \neg G x_6) \vee \neg G x_4) \tag{8.7}$$

One of the several initial sets `L` of pairs of literals is as follows:

$$\{\{G x_3, \neg G x_4\}, \tag{8.8}$$
$$\{G x_3, \neg G x_6\}, \tag{8.9}$$
$$\{G y_2, \neg G x_4\}, \tag{8.10}$$
$$\{G y_2, \neg G x_6\}, \tag{8.11}$$
$$\{F x_1 y_1, \neg F x_4 x_6\}\} \tag{8.12}$$

This selection of pairs of literals results in a ∧I-minimal proof only if one specifies that x_3 and x_4 are both to be replaced with y_2 in (8.8) and that x_3 and x_6 are both to be replaced with y_1 in (8.9). This determines that both x_3 and x_6 must be replaced with y_1 and y_2. This would be avoided if it were to be specified that x_3 and x_6 were both to be replaced with y_2 in (8.9). However, this would cause the ∧I-minimal proof, in which ∧I is applied to replace both x_6 and x_3 with y_1 and y_2 (or suitable derivates of y_1 and y_2), not to be found. By contrast, if `sub` initially specifies that both x_3 and x_6 must be replaced with y_1 and y_2, then a

criteria for restricting the number of alternative specifications `sub` without excluding any ∧I-minimal proofs. However, no such criteria are defined in the FOL-Decider for now. This is one of the reasons why an enormous number of alternative proof paths are generated in step 2 of the FOL-Decider.

To specify `sub` further, let us introduce the concept of *xx positions* of a pair of unifiable literals. Roughly speaking, identical positions in two literals forming a unifiable pair $\{L1, L2\}$ are *xx positions* iff (i) they are occupied by x variables and (ii) the occurrence of x and y variables in certain positions in $L1$ and $L2$ does not determine specific y variables that must replace x variables occurring in identical positions for the sake of unification. Thus, *xx* positions are relevant to *case 2.1* and *case 2.2*. To define *xx* positions more precisely and algorithmically, let us recall Definitions 7.3 and 7.5 of [Lampert (2019b)], which define *xx* lists as well as both *xy* and *yx* pairs:

∧I-minimal proof is found with the following D_i^*:

$$\begin{gathered}\forall x_1 \exists y_1 F x_1 y_1 \wedge \forall x_2 \forall x_5 (\forall x_7 (F x_5 x_7 \vee \neg F x_2 x_7) \vee \neg F x_5 x_2) \wedge \\ \forall x_{3_1} (\exists y_{2_1} (F x_{3_1} y_{2_1} \wedge G y_{2_1}) \vee G x_{3_1}) \wedge (\exists y_{2_2} G y_{2_2} \vee \forall x_{3_2} G x_{3_2}) \wedge \\ \forall x_{4_1} (\forall x_{6_1} (\neg F x_{4_1} x_{6_1} \vee \neg G x_{6_1}) \vee \neg G x_{4_1}) \wedge \forall x_{4_2} (\forall x_{6_2} (\neg F x_{4_2} x_{6_2} \vee \neg G x_{6_2}) \vee \neg G x_{4_2})\end{gathered} \tag{8.13}$$

In the course of applying ∧I to multiply $\forall x_3$, $\exists y_2$ is also multiplied because $\exists y_2$ is in the scope of $\forall x_3$ in (8.7). On the proof path that results in the proof of contradiction, extending `L` determines that $F x_{3_2} y_{2_2}$ is not a member of a pair of literals contained in `L`. Thus, $F x_{3_2} y_{2_2}$ is deleted from the final D_i^* (8.13). This, in turn, determines that $\exists y_{2_2}$ is not in the scope of $\forall x_{3_2}$ as a result of the scope minimization leading to (8.13). This makes it possible to generate the optimized (and optimal) prenex given in (8.25), with y_{2_2} to the left of x_{3_2}, which allows x_{3_2} to be replaced with y_{2_2} by applying ∀E. As a consequence of multiplying $\forall x_6$, $\forall x_4$ is also multiplied (cf. p. 49). In contrast to x_{6_1} and x_{6_2}, however, x_{4_1} and x_{4_2} are not replaced with different y variables in the final ∧I-minimal proof. The minimal set `L` of pairs of literals given in (8.8) - (8.12) must be extended due to the sequence of ∧I applications. This results in the following final `L`:

$$\{\{G x_{3_2}, \neg G x_{4_1}\}, \tag{8.14}$$
$$\{G x_{3_1}, \neg G x_{6_1}\}, \tag{8.15}$$
$$\{G y_{2_2}, \neg G x_{4_1}\}, \tag{8.16}$$
$$\{G y_{2_2}, \neg G x_{6_2}\}, \tag{8.17}$$
$$\{G x_{3_2}, \neg G x_{4_2}\}, \tag{8.18}$$
$$\{G y_{2_1}, \neg G x_{6_2}\}, \tag{8.19}$$
$$\{F x_1 y_1, \neg F x_{4_1} x_{6_1}\}, \tag{8.20}$$
$$\{F x_5 x_7, \neg F x_{4_2} x_{6_2}\}, \tag{8.21}$$
$$\{F x_1 y_1, \neg F x_5 x_2\}, \tag{8.22}$$
$$\{F x_{3_1} y_{2_1}, \neg F x_2 x_7\}\} \tag{8.23}$$

The following substitution list σ specifies how to replace x variables with y variables to unify all pairs of literals in (8.14) - (8.23):

$$\begin{gathered}\{\{x_1, y_{2_2}\}, \{x_2, y_1\}, \{x_5, y_{2_2}\}, \{x_7, y_{2_1}\}, \{x_{3_1}, y_1\}, \{x_{3_2}, y_{2_2}\}, \\ \{x_{4_1}, y_{2_2}\}, \{x_{4_2}, y_{2_2}\}, \{x_{6_1}, y_1\}, \{x_{6_2}, y_{2_1}\}\}\end{gathered} \tag{8.24}$$

The following prenex $\wp$ allows for these substitutions:

$$\{y_{2_2}, x_1, y_1, x_{3_1}, y_{2_1}, x_7, x_5, x_2, x_{6_1}, x_{4_1}, x_{6_2}, x_{4_2}, x_{3_2}\} \tag{8.25}$$

If one generates the prenex normal form D_i^{**} with the prenex given in (8.25) from the anti-prenex normal form given in (8.13), then all substitutions in (8.24) can be realized by applying ∀E.

Definition 7.3: The *xx lists* of a pair of connected literals $\{L1, L2\}$ are generated as follows:

(1) If an x variable ν_1 and an x variable ν_2 occur in identical positions in L1 and L2, then ν_1 and ν_2 are members of the same xx list.

(2) If the x variables ν_1 and ν_2 are members of the same xx list and if ν_2 and the x variable ν_3 are also members of the same xx list, then ν_1, ν_2 and ν_3 are all members of the same xx list.

(3) (1) and (2) are the only conditions that determine members of the same xx list.

Definition 7.5: An *xy pair* is an ordered list $\{xvar, yvar\}$ consisting of an x variable $xvar$ and a y variable $yvar$, where $xvar$ occurs in L1 in a position n and $yvar$ occurs in L2 in the same position n. A *yx pair* is an ordered list $\{yvar, xvar\}$ consisting of a y variable $yvar$ and an x variable $xvar$, where $yvar$ occurs in L1 in a position n and $xvar$ occurs in L2 in the same position n.

Based on these definitions, xx positions are defined as follows:

Definition 8.11. Let all x variables of $L1$ be subscripted with -1, and let all x variables of $L2$ be subscripted with -2. Then, identical positions in $L1$ and $L2$ are *xx positions* iff (i) they are occupied by x variables and (ii) these x variables are members of xx lists, which contain no x variables that are members of an xy or yx pair.

The condition specified in the first sentence of the above definition is important because it cannot be presupposed that the same x variable is to be replaced with the same y variable in both $L1$ and $L2$ (cf. Remark 8.13).

Note that neither other pairs of literals in the same `L` nor the order of the quantifiers in D_i^* that bind variables from $\{L1, L2\}$ are considered in the definition of xx positions.

Example 8.12. In $\{Fx_1x_1, \neg Fx_2y_1\}$, x_1 and x_2 do not occur in xx positions because x_1 must be replaced with y_1 in position 2 of Fx_1x_1 for unification. Therefore, x_1 must also be replaced with y_1 in position 1 in the same literal. Consequently, the first position in $L1$ and the first position in $L2$ are not xx positions. The occupation of identical positions in $L1$ and $L2$ by x variables is a necessary but not a sufficient condition for these positions to be xx positions.

xx positions can be occupied by the same x variable, as in the case of, e.g., $\{Fx_1, \neg Fx_1\}$.

Example 8.13. It might well be that the same x variable is to be replaced with different y variables in $L1$ and $L2$. For example, in $\{Fx_1y_1x_3, \neg Fx_2x_1y_1\}$, x_1 must be replaced only with y_1 in $L2$; in $L1$, however, x_1 occurs in an xx position. Therefore, the pair of literals does not dictate that x_1 must be replaced with y_1 in the first position of $L1$. Thus, the FOL-Decider proves that the formula $\forall x_1 \exists y_1(\forall x_3 Fx_1y_1x_3 \wedge \forall x_2 \neg Fx_2x_1y_1)$ is refutable by means of a single application of $\wedge$I to replace x_{1_1} in $Fx_{1_1}y_{1_1}x_{3_1}$ with y_0 and x_{1_2} in $\neg Fx_{2_2}x_{1_2}y_{1_2}$ with y_{1_1} in the resulting D_i^{**}.[22]

Definition 8.14. A *specification* `sub` *of substitutions* specifies a set of substitutions of x variables in all positions in literals in `L` such that all single pairs of literals are unified.

[22]The resulting D_i^{**} is

$$\forall x_{1_1} \exists y_{1_1} \forall x_{1_2} \exists y_{1_2} \forall x_{3_1} \forall x_{2_2}(Fx_{1_1}y_{1_1}x_{3_1} \wedge \neg Fx_{2_2}x_{1_2}y_{1_2}) \tag{8.26}$$

This D_i^{**} is generated from the following purged* D_i^*:

$$\forall x_{1_1} \exists y_{1_1} \forall x_{3_1} Fx_{1_1}y_{1_1}x_{3_1} \wedge \forall x_{1_2} \exists y_{1_2} \forall x_{2_2} \neg Fx_{2_2}x_{1_2}y_{1_2} \tag{8.27}$$

The FOL-Decider combines L and sub into a list denoted by sub/L, in which each pair of literals from L is preceded by a list that specifies how its x variables are to be substituted according to sub. Each sub for a single pair of literals is a list of three lists. The first list specifies which y variables must replace x variables due to unification requirements for that single pair of literals (cf. *case 1* on p. 43). The second and the third list concern xx positions. The second list specifies how x variables must be replaced with y variables as dictated by substitutions in other pairs of literals in L (cf. *case 2.1* on p. 43). The third list specifies x variables without specific substitution requirements; it simply specifies which of them must be replaced with the same y variable (cf. *case 2.2* on p. 43). Each of these three lists concerns the substitution requirements for x variables in certain positions in the corresponding pair of literals. These lists and the corresponding pair of literals together dictate how the variable in each position that is occupied by an x variable must be substituted. The generation of sub/L and its modifications and extensions all play crucial roles in the FOL-Decider.

Example 8.15. The sub/L for (8.2) - (8.5) from Example 8.9 that corresponds to specification (1) for the xx positions of x_1 and x_2 in (8.4) and (8.5) (cf. p. 44) is as follows:

$$\{\{\{\{\{x_1, y_2\}\}, \{\}, \{\}\}, \{Fx_1, \neg Fy_2\}\}, \tag{8.28}$$

$$\{\{\{\{x_2, y_1\}, \{x_5, y_1\}\}, \{\}, \{\}\}, \{Gy_1y_1, \neg Gx_2x_5\}\}, \tag{8.29}$$

$$\{\{\{\{x_3, y_2\}\}, \{\{x_1, x_2, y_1\}, \{x_4, x_5, y_1\}, \{\}\}, \{Hx_1x_3x_4, \neg Hx_2y_2x_5\}\}, \tag{8.30}$$

$$\{\{\{\{x_3, y_2\}\}, \{\{x_1, x_2, y_1\}\}, \{\}\}, \{Jx_1x_3, \neg Jx_2y_2\}\}\} \tag{8.31}$$

Remark 8.16. The substitutions of x variables in xx positions that are specified in sub depend on the pairs of literals from L. As discussed on p. 48f., the replacement of x variables with y_0 to satisfy condition $\mathtt{P}_{\wp}$ additionally depends on optimized prenexes $\wp$ and is also considered in sub.

The FOL-Decider generates all possible alternative specifications sub for the unification of the pairs of literals in L.

Definition 8.17. A *substitution list* σ for a set L of unifiable pairs of literals is a list consisting of lists that specify, for each x variable from L, which y variables must replace it according to sub for the unification of all pairs of literals in L.

Remark 8.18. The FOL-Decider generates substitution lists σ from sub.

Example 8.19. The substitution list σ for the sub given in (8.28) to (8.31) is

$$\sigma: \quad \{\{x_1, y_1.y_2\}, \{x_2, y_1\}, \{x_3, y_1\}, \{x_4, y_1\}, \{x_5, y_1\}\} \tag{8.32}$$

Strictly speaking, the FOL-Decider differentiates between substitution lists, which contain only x variables that are to be replaced with at least one y variable according to sub, and final substitution lists, which contain all x variables from sub. This distinction concerns x variables ν for which no specific y variables are specified in sub to replace them (cf. *case 2.2*, p. 43). Final substitution lists include lists of the form $\{\nu, y_0\}$ for these x variables. However, the final substitution lists are a rather insignificant detail of the FOL-Decider; they are relevant only in the case that a proof is found. Thus, I abstain from distinguishing

Note that ∃M (cf. p. 10 in [Lampert (2019b)]) is not applied by default in step 1 of the FOL-Decider (cf. footnote 19 above and footnote 9 in [Lampert (2019b)]). Therefore, the FOL-Decider proves that $\forall x_1 \exists y_1 (\forall x_3 Fx_1y_1x_3 \wedge \forall x_2 \neg Fx_2x_1y_1)$ is refutable by means of a single application of ∧I.

between these two kinds of substitution lists in the following. Unless the substitution lists σ at hand are those returned by the FOL-Decider in the case that a $\wedge$I-minimal proof is found, I refer to substitution lists σ in the sense of the first meaning.[23]

Definition 8.20. A substitution list σ is *unambiguous* iff each partial list in σ is of length 2 (i.e., contains exactly one y variable).

Remark 8.21. σ is unambiguous iff `sub` is unambiguous (cf. p. 41).

As soon as σ is unambiguous apart from the introduction of y_0 into `sub` to satisfy $\mathtt{P}_\wp$, `sub` must satisfy the *Prenex Principle* $\mathtt{P}_\wp$.

Prenex Principle ($\mathtt{P}_\wp$). *Each x variable ν of an optimized prenex $\wp$ appears to the right of at least one y variable that is to be substituted for ν according to the specification* `sub` *of substitutions.*

$\mathtt{P}_\wp$ ensures that an explicit contradiction D_i^{***} can be deduced from D_i given that σ is unambiguous and $\mathtt{P}_\wp$ is satisfied. In this case, the anti-prenex normal form D_i^* that is deduced from D_i via $\wedge$I applications can be purged* and converted into a prenex normal form D_i^{**} with an optimized prenex $\wp$ that allows $\forall$E to be applied to replace all x variables with appropriate y variables according to σ. Because σ is generated from `sub`, this is sufficient to unify all literals from `L`. Consequently, any disjunct of the DNF matrix of D_i^{***} contains an explicit contradiction. Therefore, the unambiguity of σ is a sufficient criterion for a $\wedge$I-minimal proof once $\mathtt{P}_\wp$ is considered.

Thus, $\wedge$I must be applied only if σ is not unambiguous. If $\mathtt{P}_\wp$ is not satisfied for an x variable ν, then ν must additionally be replaced with y_0 in order to satisfy $\mathtt{P}_\wp$. By definition, $\exists y_0$ precedes any optimized prenex $\wp$. A specification `sub` of substitutions considers $\mathtt{P}_\wp$ as soon as σ is unambiguous apart from satisfying $\mathtt{P}_\wp$.[24] The FOL-Decider considers all combinatorial possibilities for the introduction of y_0 to satisfy $\mathtt{P}_\wp$ (for an illustration, cf. Example 8.28).[25] Thus, introducing y_0 for the sake of satisfying $\mathtt{P}_\wp$ results in further alternative specifications `sub` and, consequently, alternative substitution lists σ, thus necessitating alternative proof paths. These alternative proof paths consider all combinatorial variants of minimal sets `L` of unifiable pairs of literals, optimized prenexes $\wp$ and alternative substitution specifications `sub`. If the substitution of x variables with y_0 is no longer necessary to satisfy $\mathtt{P}_\wp$ subsequent to $\wedge$I applications, this is also considered via corresponding modifications of `sub` (cf. *exception 2* on p. 54).

An optimized prenex $\wp$ is *optimal* iff σ is unambiguous given that $\mathtt{P}_\wp$ is considered.

---

[23]The algorithm for generating substitution lists σ concerns the substitution of x variables from a *minimal set* of unifiable pairs of literals in step 2 of the FOL-Decider. This algorithm must be distinguished from Algorithm 9.1 described in section 9 of [Lampert (2019b)], which generates *maximal* substitution lists for a *single* pair of connected literals. The generation of substitution lists σ in step 2 of the FOL-Decider serves not for deciding whether connected literals are unifiable but rather for deciding whether further $\wedge$I applications are necessary in the search for a $\wedge$I-minimal proof along a given proof path.

[24]If $\mathtt{P}_\wp$ were to be considered in previous proof steps with ambiguous substitution lists σ, the necessity to introduce substitutions of x variables with y_0 would not be kept to a minimum. Example 8.10 in footnote 21 on p. 44 illustrates this. Given an optimized prenex in which x_3 is to the left of y_1 and y_2, x_3 would have to be additionally replaced with y_0. However, this additional substitution is not necessary for the $\wedge$I-minimal proof described in Example 8.10.

[25]Again, it might be possible to define criteria to restrict the possible combinations without excluding any $\wedge$I-minimal proofs. However, no such criteria are defined in the FOL-Decider for now.

Definition 8.22. An optimized prenex $\wp$ is *optimal* iff each x variable μ in $\wp$ is to the right of the y variable ν that is to replace it according to the unambiguous substitution list σ.

8.3. **$\wedge$I Applications.** Applications of $\wedge$I are necessary only if σ is ambiguous. In this case, at least one x variable μ in `sub` must be multiplied to enable the substitution of $\mu_1, \ldots, \mu_n$ with n different y variables.

Definition 8.23. An x variable μ *is multiplied* iff the expression $\forall\mu A(\mu)$ in D_i^* is multiplied by applying $\wedge$I.

To avoid the application of DIS2 when generating anti-prenex normal forms in step 2 of the FOL-Decider, it is, strictly speaking, not only the expression $\forall\mu A(\mu)$ that is multiplied; instead, what is multiplied is the entire expression that contains all universal quantifiers and occurrences of $\vee$ preceding $\forall\mu A(\mu)$ in the logical hierarchy of D_i^* up to the next occurrence of some existential quantifier or $\wedge$. This multiplied expression will always be preceded by a universal quantifier (cf., e.g., Example 8.10 in footnote 21: to multiply x_6, the expression preceded by $\forall x_4$ is multiplied, as seen in the last line in (8.13)). For simplicity, I will still use $\forall\mu A(\mu)$ to refer to the expression that is multiplied, which may be preceded by universal quantifiers other than $\forall\mu$.

$\forall\mu A(\mu)$ is multiplied $n-1$ times by applying $\wedge$I to generate n conjuncts such that μ_i is replaced with the i-th of the n y variables with which μ must be replaced according to σ. To distinguish among different $\wedge$I applications for the multiplication of different x variables, I refer to all of the $n-1$ applications of $\wedge$I that are necessary to multiply *one* x variable μ as "*one* $\wedge$I application". The steps along a proof path are numbered with respect to the $\wedge$I applications that are performed to multiply x variables. Although x variables ν ($\nu \neq \mu$) that are bound by universal quantifiers occurring in $\forall\mu A(\mu)$ are also multiplied when multiplying $\forall\mu A(\mu)$, only μ is "*the* multiplied x variable", because the purpose of applying $\wedge$I is to replace instances of μ in different conjuncts with different y variables.

Definition 8.24. *One* $\wedge$I application consists of the replacement of the universally quantified expression $\forall\mu A(\mu)$ with a conjunction of n conjuncts $\forall\mu_1 A(\mu_1), \ldots, \forall\mu_n A(\mu_n)$, in which all variables bound by quantifiers of the i-th conjunct are subscripted with i for rectification.

In the course of the $\wedge$I application, `L`, $\wp$, and `sub` must be modified and extended. In the following, I distinguish `L` and `sub` prior to a $\wedge$I application from their *modifications* $\texttt{L}'$ and $\texttt{sub}'$ directly subsequent to the $\wedge$I application and their *extensions* $\texttt{L}''$ and $\texttt{sub}''$ prior to possible further $\wedge$I applications. The resulting $\texttt{L}''$ and $\texttt{sub}''$ subsequent to a given $\wedge$I application are identical to the `L` and `sub` prior to the next $\wedge$I application. In the case of $\wp$, I distinguish $\wp$ prior to a $\wedge$I application from the subsequent *modified and extended* $\wp'$ prior to possible further $\wedge$I applications. However, I will speak of `L`, $\wp$, and `sub` in general if I am not considering their modifications or extensions.

Subsequent to a $\wedge$I application, the FOL-Decider first modifies `L` (resulting in $\texttt{L}'$) and `sub` (resulting in $\texttt{sub}'$). However, if $\forall\mu A(\mu)$ contains $\vee$, then $\texttt{L}'$ must also be extended subsequent to the $\wedge$I application (cf. section 8.4 for details). This extension results in $\texttt{L}''$ and is considered only after the modification of `L` and `sub` to $\texttt{L}'$ and $\texttt{sub}'$. During the generation of `sub`, $\texttt{sub}'$ and $\texttt{sub}''$, the substitutions of x variables in positions other than xx positions are specified prior to those in xx positions, as the latter substitutions depend on the former. However, I omit these details in the following. $\wp$ is modified and extended (resulting in $\wp'$) only after $\texttt{L}''$ is generated because it refers to $\texttt{L}''$. Finally, $\texttt{sub}'$ is extended to

$\mathtt{sub}''$; this extension is performed last because it depends on $\mathtt{L}''$ and $\wp'$. These modifications and extensions are explained in this and the following section.

Subsequent to the application of $\wedge$I, the index depths of variables bound by quantifiers occurring in $\forall\mu A(\mu)$ are increased by 1. The modification of $\mathtt{L}$, $\mathtt{sub}$ and $\wp$ basically concerns the replacement of "ancestors" with "descendants".

Definition 8.25. Let v be a variable bound by a quantifier in $\forall\mu A(\mu)$, and let v_1 to v_n denote the variables resulting from the $\wedge$I application. Then, the variables $v_1, \ldots, v_n$ are *descendants of* v, and v is their *ancestor*.

$\mathtt{L}'$ and $\mathtt{sub}'$ consider the replacement of ancestors with descendants.

The FOL-Decider orders the y variables with which μ must be replaced according to σ in a list `yvars` that begins with y_0. μ_i in the i-th conjunct must then be replaced with the i-th y variable from `yvars`. Thus, the corresponding literals in $\mathtt{L}$, in which μ must be replaced with the i-th y variable from `yvars`, are replaced with the literals from the i-th conjunct in D_i^*. Consequently, all variables of these literals that are bound by quantifiers occurring in $\forall\mu_i A(\mu_i)$ are subscripted with i in $\mathtt{L}'$. The substitutions in $\mathtt{sub}$ are adjusted likewise to obtain the modified $\mathtt{sub}'$.

In addition to literals containing μ, literals in $\mathtt{L}$ that do not contain μ but do contain variables bound by quantifiers occurring in $\forall\mu A(\mu)$ must be modified subsequent to the $\wedge$I application. All combinatorial possibilities for replacing literals in $\mathtt{L}$ that do not contain μ but do contain variables bound by quantifiers occurring in $\forall\mu A(\mu)$ with literals from different conjuncts of the result of the $\wedge$I application are realized in alternative sets $\mathtt{L}'$ and considered in the respective specifications $\mathtt{sub}'$. The only requirements are that the pairs of literals still be unifiable and that the selection of literals does not violate `FP`. Alternative selections of literals from the n conjuncts give rise to alternative proof paths.

Furthermore, all alternative ways of subscripting a y variable ν that replaces x variables in xx positions according to $\mathtt{sub}$ and that is bound by an existential quantifier occurring in $\forall\mu A(\mu)$ are realized in alternative substitution specifications $\mathtt{sub}'$. Such alternatives arise in the case that the corresponding x variables must be replaced with different descendants ν_j and ν_k ($j, k \in \mathbb{N}$ and $\leq n$, $j \neq k$) of the ancestor y variable ν in $\mathtt{sub}'$ in positions that are not xx positions. It might be possible to define criteria to restrict the number of alternative sets $\mathtt{L}'$ and alternative specifications $\mathtt{sub}'$ without excluding $\wedge$I-minimal proofs. However, no such criteria are defined in the FOL-Decider for now. This is another reason why an enormous number of alternative proof paths are generated in step 2 of the FOL-Decider.

The differences between $\mathtt{sub}$ and $\mathtt{sub}'$ described so far concern the replacement of ancestors with descendants. However, there is one additional difference (= *exception 1*). Let ν ($\nu \neq \mu$) be an x variable occurring in $\forall\mu A(\mu)$. Let ν be replaced with some y variable ρ in some position that is not an xx position (*case 1* on p. 43), and consequently, let ν be replaced with ρ in some other position that is an xx position (*case 2.1* on p. 43). Since ν is multiplied in the course of the $\wedge$I application, it may well be that its descendant ν_i no longer needs to be replaced with ρ (or some descendant ρ_j) in some position that is not an xx position due to $\mathtt{L}'$. Consequently, there is also no need to replace ν_i with ρ (or some descendant ρ_j) in xx positions, as long as this is also not required due to other x variables that share some xx position with ν_i. This possibility is considered when modifying $\mathtt{sub}$, to the effect that the requirement to replace ν_i in xx positions with ρ (or some descendant ρ_j) is discarded in $\mathtt{sub}'$. This keeps the number of required substitutions and, consequently, the number of necessary $\wedge$I applications low.

In addition to $\mathtt{L}$ and $\mathtt{sub}$, $\wp$ must also be modified and extended subsequent to $\wedge$I applications. First of all, this is required because quantifiers from $\forall\mu A(\mu)$ are multiplied. Furthermore, additional quantifiers may need to be considered in the course of extending $\mathtt{L}'$ to $\mathtt{L}''$ (cf. section 8.4 for details). In the following, however, I abandon the term "modification of $\wp$" and simply speak of "extensions of $\wp$", including the replacement of ancestors with their descendants. Before explaining extensions of $\mathtt{L}'$, I specify the *Principle of Prenex Extension* $\mathtt{PE}\wp$ because the principle that guides the extension of $\wp$ depends simply on the generation of all optimized prenexes $\wp+$ involving the variables in $\mathtt{L}''$ (however $\mathtt{L}''$ is generated). From all these prenexes $\wp+$, the FOL-Decider selects only those prenexes $\wp'$ that satisfy the following principle:

Principle of Prenex Extension ($\mathtt{PE}\wp$). *Prenex extensions $\wp'$ must maintain the possibilities provided by $\wp$ for replacing x variables with y variables by means of $\forall E$.*

The requirement of *maintaining* substitution possibilities is specified by the following three conditions:

(1) A y variable ν must not appear to the right of an x variable μ in $\wp'$ if ν is to the left of μ in $\wp$ unless $\exists\nu$ is in the scope of $\forall\mu$ in D_i^* after the extension of $\mathtt{L}'$ to $\mathtt{L}''$.[26]

(2) For each ancestor v occurring in $\wp$, generate the set of its descendants that occur in $\wp+$. Let k be the number of resulting non-empty sets of descendants, and let r_1 to r_k be their cardinalities. For these sets, generate their Cartesian product. Each member of the Cartesian product contains exactly one descendant for each ancestor. For each of the $r_1 \cdot \ldots \cdot r_k$ members m_i of the Cartesian product, generate the sequence $\wp^*$ from $\wp+$ by replacing the descendants of m_i with their ancestor. $\wp+$ is a prenex $\wp'$ that satisfies $\mathtt{PE}\wp$ only if at least one of the resulting $r_1 \cdot \ldots \cdot r_k$ prenexes $\wp^*$ satisfies condition (1) compared to $\wp$. This test ensures that for each ancestor in $\wp$, at least one descendant takes over its relative position in $\wp'$.

(3) Each descendant μ_i of the multiplied x variable μ must be to the right of the y variable with which μ_i must be replaced according to $\mathtt{sub}'$, if $\mathtt{P}_\wp$ is satisfied prior to the $\wedge$I multiplication.[27]

[26]This exception is necessary because optimized prenexes are generated in relation to *purged** D_i^*. The following example illustrates this:

$$\forall x_1\exists y_1(\exists y_2(\exists y_3(Gy_1y_3 \wedge Gy_2y_3) \wedge \exists y_4\forall x_3\neg Hy_4y_2x_3) \wedge \forall x_4 Hy_1x_4x_1) \wedge$$
$$\forall x_2(\forall x_5\forall x_8 Hx_5x_2x_8 \vee \forall x_6\forall x_9\neg Hx_2x_6x_9 \vee \forall x_7\neg Gx_2x_7) \tag{8.33}$$

One of the two minimal sets $\mathtt{L}$ of (8.33) is $\{\{Gy_2y_3,\ \neg Gx_2x_7\},\ \{Hx_5x_2x_8,\ \neg Hy_4y_2x_3\},\ \{Hy_1x_4x_1,\ \neg Hx_2x_6x_9\}\}$. With regard to this $\mathtt{L}$, Gy_1y_3 is deleted during the process of purging. This results in the following optimized prenex $\wp$:

$$\{y_2, y_3, y_4, x_1, y_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\} \tag{8.34}$$

However, the second conjunct of (8.33) must be multiplied to prove the refutability of (8.33) because x_2 must be replaced with both y_1 and y_2 in the considered $\mathtt{L}$. In the course of the $\wedge$I application, $\mathtt{L}'$ must be extended by $\{Gy_1y_3, \neg Gx_{2_2}x_{7_2}\}$. From this, it follows that Gy_1y_3 is now part of D_i^*. Consequently, y_2, y_3, and y_4 cannot be to the left of x_1 in $\wp'$.

[27]The if-clause "if $\mathtt{P}_\wp$ ..." is necessary because in the search for proofs, x variables are multiplied without considering the *Prenex Condition* $\mathtt{P}_\wp$ as long as σ is ambiguous (cf. p. 48 and footnote 24). This is illustrated in the following example.

Example 8.26. Let D_i be given as follows:

$$\forall x_1\exists y_1(\forall x_2\forall x_6 Fx_2x_1y_1x_6 \wedge \forall x_3\forall x_7 Gx_3x_1y_1x_7 \wedge \forall x_4\forall x_8 Hx_4x_1y_1x_8 \wedge$$
$$\forall x_5(\exists y_2\forall x_9\neg Fx_5y_2x_9y_1 \vee \exists y_3\forall x_{10}\neg Gx_5y_3x_{10}y_1 \vee \forall x_{11}\forall x_{12}\neg Hx_5x_{11}x_{12}y_1)) \tag{8.35}$$

$\mathtt{PE}\wp$ ensures that the necessity to replace x variables additionally with y_0 is kept rare by the extension of $\wp$. Furthermore, condition (3) of $\mathtt{PE}\wp$ ensures that all replacements of μ_1 to μ_n with the respective y variables from $\mathtt{sub}'$ can be achieved by applying $\forall$E to a corresponding prenex normal form D_i^{**} given that $\mathtt{P}_\wp$ is satisfied prior to the $\wedge$I application. This was the stated purpose of the $\wedge$I application.

Remark 8.27. $\mathtt{PE}\wp$ is not a necessary principle for a $\wedge$I-minimal proof search. For a decision procedure for D_i based on a $\wedge$I-minimal proof search, it is sufficient to assume the finiteness of the combinatorial possibilities for generating prenexes (cf. section 11).

Alternative possibilities for extending $\wp$ to $\wp'$ are considered on alternative proof paths. The modifications and extensions of $\mathtt{L}$, $\wp$, and $\mathtt{sub}$ consider all alternatives that might result in a $\wedge$I-minimal proof.

Example 8.28. This example illustrates how a $\wedge$I-minimal proof is found when considering substitutions with y_0 to satisfy condition $\mathtt{P}_\wp$. Extensions of $\mathtt{L}'$ (cf. section 8.4) need not be considered in this example.

Let D_i be given as follows:

$$\forall x_1 \exists y_1 F x_1 y_1 \wedge \forall x_2 (\forall x_3 \neg F x_2 x_3 \vee \forall x_4 \neg F x_4 x_2) \tag{8.42}$$

There is exactly one minimal set $\mathtt{L}$ of unifiable pairs of literals in the case of (8.42):

$$\{\{F x_1 y_1, \neg F x_2 x_3\}, \tag{8.43}$$
$$\{F x_1 y_1, \neg F x_4 x_2\}\} \tag{8.44}$$

Also, there is exactly one optimized prenex $\wp$:

The FOL-Decider finds a $\wedge$I-minimal proof in which the final D_i^* is as follows:

$$\begin{gathered}\forall x_{1_1} \exists y_{1_1} \forall x_{2_1} \forall x_{6_1} F x_{2_1} x_{1_1} y_{1_1} x_{6_1} \wedge \\ \forall x_{1_{2_1}} \exists y_{1_{2_1}} (\forall x_{4_{2_1}} \forall x_{8_{2_1}} H x_{4_{2_1}} x_{1_{2_1}} y_{1_{2_1}} x_{8_{2_1}} \wedge \\ \forall x_{5_{2_1}} (\exists y_{2_{2_1}} \forall x_{9_{2_1}} \neg F x_{5_{2_1}} y_{2_{2_1}} x_{9_{2_1}} y_{1_{2_1}} \vee \\ \exists y_{3_{2_1}} \forall x_{10_{2_1}} \neg G x_{5_{2_1}} y_{3_{2_1}} x_{10_{2_1}} y_{1_{2_1}} \vee \\ \forall x_{11_{2_1}} \forall x_{12_{2_1}} \neg H x_{5_{2_1}} x_{11_{2_1}} x_{12_{2_1}} y_{1_{2_1}})) \wedge \\ \forall x_{1_{2_2}} \exists y_{1_{2_2}} \forall x_{3_{2_2}} \forall x_{7_{2_2}} G x_{3_{2_2}} x_{1_{2_2}} y_{1_{2_2}} x_{7_{2_2}}\end{gathered} \tag{8.36}$$

(8.36) results from a first application of $\wedge$I to replace descendants of x_1 with descendants of y_2 and y_3 and a second application of $\wedge$I to additionally replace a descendant of x_{1_2} with y_0 in order to satisfy $\mathtt{P}_\wp$. This $\wedge$I application would be omitted if one were to neglect the if-clause in condition (3). The $\wedge$I-minimal proof of contradiction is based on the following final $\mathtt{L}$, σ, and $\wp$.

$\mathtt{L}$:

$$\{\{F x_{2_1} x_{1_1} y_{1_1} x_{6_1}, \neg F x_{5_{2_1}} y_{2_{2_1}} x_{9_{2_1}} y_{1_{2_1}}\}, \tag{8.37}$$
$$\{H x_{4_{2_1}} x_{1_{2_1}} y_{1_{2_1}} x_{8_{2_1}}, \neg H x_{5_{2_1}} x_{11_{2_1}} x_{12_{2_1}} y_{1_{2_1}}\}, \tag{8.38}$$
$$\{G x_{3_{2_2}} x_{1_{2_2}} y_{1_{2_2}} x_{7_{2_2}}, \neg G x_{5_{2_1}} y_{3_{2_1}} x_{10_{2_1}} y_{1_{2_1}}\}\} \tag{8.39}$$

σ:

$$\begin{gathered}\{\{x_{1_1}, y_{2_{2_1}}\}, \{x_{2_1}, y_0\}, \{x_{6_1}, y_{1_{2_1}}\}, \{x_{1_{2_1}}, y_0\}, \{x_{1_{2_2}}, y_{3_{2_1}}\}, \{x_{3_{2_2}}, y_0\}, \{x_{4_{2_1}}, y_0\}, \\ \{x_{5_{2_1}}, y_0\}, \{x_{7_{2_2}}, y_{1_{2_1}}\}, \{x_{8_{2_1}}, y_{1_{2_1}}\}, \{x_{9_{2_1}}, y_{1_1}\}, \{x_{10_{2_1}}, y_{1_{2_2}}\}, \{x_{11_{2_1}}, y_0\}, \{x_{12_{2_1}}, y_{1_{2_1}}\}\}\end{gathered} \tag{8.40}$$

$\wp$:

$$\begin{gathered}\{y_0, x_{1_{2_1}}, y_{1_{2_1}}, x_{5_{2_1}}, y_{2_{2_1}}, y_{3_{2_1}}, x_{1_1}, y_{1_1}, x_{1_{2_2}}, y_{1_{2_2}}, \\ x_{2_1}, x_{6_1}, x_{8_{2_1}}, x_{4_{2_1}}, x_{9_{2_1}}, x_{10_{2_1}}, x_{12_{2_1}}, x_{11_{2_1}}, x_{3_{2_2}}, x_{7_{2_2}}\}\end{gathered} \tag{8.41}$$

$$\{x_1, y_1, x_2, x_3\} \tag{8.45}$$

x_3 must be replaced with y_1 in (8.43), and x_2 must be replaced with y_1 in (8.44). x_1 and x_2 must be replaced with the same y variable in (8.43), and x_1 and x_4 must be replaced with the same y variable in (8.44). If $\mathtt{P}_{\wp}$ is not considered, then x_1 and, consequently, x_4 must be replaced only with y_1 because x_2 must be replaced with y_1 in (8.44). Thus, σ is unambiguous as long as $\mathtt{P}_{\wp}$ is not considered. However, $\mathtt{P}_{\wp}$ is not satisfied because y_1 is not to the left of x_1 in (8.45). $\forall$E cannot be applied to replace x_1 with y_1 in the corresponding prenex normal form with the optimized prenex given in (8.45). Therefore, x_1 must additionally be replaced with y_0 in order to satisfy $\mathtt{P}_{\wp}$. Either (1) x_1 and, consequently, x_2 must be replaced with y_0 in (8.43); (2) x_1 and, consequently, x_4 must be replaced with y_0 in (8.44); or (3) both. For completeness, I also note the special case (4), which specifies that x_1 must be replaced with y_0 in addition to y_1 but not in the xx positions in (8.43) and (8.44). This case is also considered in the FOL-Decider. Cases (1) to (4) give rise to the following 4 substitution lists, $\sigma(1)$ to $\sigma(4)$:

(1) $\sigma(1) = \{\{x_1, y_0\}, \{x_2, y_0, y_1\}, \{x_3, y_1\}, \{x_4, y_0\}\}$,
(2) $\sigma(2) = \{\{x_1, y_0, y_1\}, \{x_2, y_1\}, \{x_3, y_1\}, \{x_4, y_0\}\}$,
(3) $\sigma(3) = \{\{x_1, y_0, y_1\}, \{x_2, y_0, y_1\}, \{x_3, y_1\}, \{x_4, y_1\}\}$,
(4) $\sigma(4) = \{\{x_1, y_0, y_1\}, \{x_2, y_1\}, \{x_3, y_1\}, \{x_4, y_1\}\}$.

These 4 variants correspond to 4 alternative proof paths. The special case (4), with $\sigma(4)$, does not result in a $\wedge$I-minimal proof because x_{1_1} is not replaced with y_0 in $Fx_{1_1}y_{1_1}$. Instead, x_{1_2} is replaced with y_{1_1} in $Fx_{1_2}y_{1_2}$. Consequently, $\mathtt{L}'$ does not contain any literal from the conjunct $\forall x_{1_1}\exists y_{1_1}Fx_{1_1}y_{1_1}$ that results from multiplying $\forall x_1\exists y_1Fx_1y_1$. This contradicts the *Fundamental Principle* FP (cf. p. 42). Therefore, this proof path terminates because it contradicts a principle of the $\wedge$I-minimal proof search. The FOL-Decider first realizes all combinatorial possibilities (1) to (4) and, consequently, $\sigma(1)$ to $\sigma(4)$. The proof path based on $\sigma(4)$ is then abandoned subsequent to the $\wedge$I application because it violates FP.

In the cases of variants (1) and (3), $\sigma(1)$ and $\sigma(3)$ cause it to be necessary to multiply x_2. This does not result in a $\wedge$I-minimal proof. Therefore, let us consider only variant (2), with $\sigma(2)$. According to $\sigma(2)$, only x_1 must be multiplied. This results in the following D_i^*:

$$\forall x_{1_1}\exists y_{1_1}Fx_{1_1}y_{1_1} \wedge \forall x_{1_2}\exists y_{1_2}Fx_{1_2}y_{1_2} \wedge \forall x_2(\forall x_3\neg Fx_2x_3 \vee \forall x_4\neg Fx_4x_2) \tag{8.46}$$

A $\wedge$I-minimal proof is found if one replaces x_{1_1} with y_0 to unify $\{Fx_{1_1}y_{1_1}, \neg Fx_4x_2\}$ and replaces x_{1_2} with y_{1_1} to unify $\{Fx_{1_2}y_{1_2}, \neg Fx_2x_3\}$. This proof rests on the unification of the following pairs of literals from (8.46):

$$\{\{Fx_{1_1}y_{1_1}, \neg Fx_4x_2\}, \{Fx_{1_2}y_{1_2}, \neg Fx_2x_3\}\} \tag{8.47}$$

It also rests on the following substitution list σ:

$$\{\{x_{1_1}, y_0\}, \{x_{1_2}, y_{1_1}\}, \{x_2, y_{1_1}\}, \{x_3, y_{1_2}\}, \{x_4, y_0\}\} \tag{8.48}$$

The corresponding optimal prenex $\wp$ is as follows:

$$\{y_0, x_{1_1}, y_{1_1}, x_{1_2}, y_{1_2}, x_2, x_3, x_4\} \tag{8.49}$$

If one generates the corresponding optimal prenex form D_i^{**} from (8.46) with the optimal prenex given in (8.49), then one can deduce an explicit contradiction D_i^{***} by applying $\forall$E.

8.4. **Extending $\mathtt{L}'$ and $\mathtt{sub}'$.** If the multiplied expression $\forall\mu A(\mu)$ contains $\vee$, then $\mathtt{L}'$, $\wp$, and $\mathtt{sub}'$ must be extended.

According to conditions (i) and (ii) in Definition 8.4, the following two principles hold for the extension of $\mathtt{L}'$.

$\bot$-Principle ($\bot\mathtt{P}$). *$\mathtt{L}''$ must satisfy the condition that each disjunct of the DNF matrix of D_i^* contains two conjuncts, A and $\neg A$, given that all pairs of literals in $\mathtt{L}''$ are unified.*

Minimal $\mathtt{L}$-Principle ($\mathtt{M_L P}$). *$\mathtt{L}''$ is* minimally sufficient[28] *to satisfy $\bot\mathtt{P}$.*

Both principles also hold for any $\mathtt{L}$ prior to any application of $\wedge$I. $\mathtt{M_L P}$ avoids unnecessary $\wedge$I applications that may violate the *Fundamental Principle* FP of the $\wedge$I-minimal proof search strategy.

In addition to applying the *Fundamental Principle* FP, the FOL-Decider restricts the selection of literals in $\mathtt{L}$ in accordance with the following principle for an effective $\wedge$I-minimal proof search:

Principle of $\mathtt{L}$-Extension (PEL). *All literals from $\mathtt{L}'$ must be maintained when extending $\mathtt{L}'$ to $\mathtt{L}''$. Through all modifications and extensions, literals from $\mathtt{L}$, once they are members of $\mathtt{L}$, are modified only by increasing the index depths of their variables.*

This principle ensures that any selection of a literal in the course of the proof search must be necessary for the $\wedge$I-minimal proof. It follows that along a given proof path, the size of $\mathtt{L}$ and the number of selected conjuncts can only increase.

The FOL-Decider generates the initial $\mathtt{L}$ and any extensions $\mathtt{L}''$ in accordance with the principles FP, $\bot\mathtt{P}$, $\mathtt{M_L P}$, and PEL.

Similar to the principles $\mathtt{PE}_\wp$ and PEL, the substitutions in $\mathtt{sub}'$ must also be maintained in $\mathtt{sub}''$. An exception can be found in certain substitutions of x variables μ with y_0. Let μ be an x variable that must be replaced with y_0 according to $\mathtt{sub}'$ in order to satisfy the *Prenex Condition* $\mathtt{P}_\wp$ with respect to $\wp$. Let $\mathtt{L}'$ be extended such that μ must additionally be replaced with a y variable ν that is to the left of μ according to $\wp'$. In this case, ν can take over the role of y_0, meaning that μ no longer needs to be replaced with y_0. This is *exception 2* of the maintenance principle for the specification of substitutions sub. The FOL-Decider considers this exception and computes its consequences for the substitutions of x variables in xx positions. Like *exception 1* (cf. p. 50), *exception 2* minimizes the length of the partial list in σ, and therefore the need to multiply x variables, without excluding any $\wedge$I-minimal proofs.

Another, final, exception corresponds to substitutions at xx positions in pairs of literals in $\mathtt{L}'$. If x variables in these positions must be replaced with further y variables due to the extension of $\mathtt{L}'$ to $\mathtt{L}''$, then *additional* alternative specifications $\mathtt{sub}''$ must be generated that consider alternative substitutions of x variables in xx positions in pairs of literals from $\mathtt{L}'$

[28]This term is defined as follows:

Definition 8.29. A set S is minimally sufficient to satisfy a certain condition C iff no proper subset of S is sufficient to satisfy C.

(*exception 3*). Thus, not all replacements of x variables with y variables in xx positions are maintained in these *additional* specifications $\mathtt{sub}''$. *All* alternative substitution possibilities are considered on alternative proof paths to ensure that a $\wedge$I-minimal proof path is found if one exists.

Principle of sub-Extension (PEsub). *Apart from* exception 1 *as described on p. 50 and* exceptions 2 *and* 3 *as introduced above, all substitutions must be maintained subsequent to* $\wedge I$ *applications: through all modifications and extensions, variables from* sub *are modified only by increasing their index depths, apart from the mentioned exceptions.*

The mentioned *exceptions* are justified by the intent to find a $\wedge$I-minimal proof, if it exists, without generating superfluous proof paths.

I summarize the maintenance principles for extending $\mathtt{L}'$, $\wp$, and $\mathtt{sub}/\mathtt{sub}'$ into the following single principle:

Principle of Extension (PE). L, $\mathtt{L}'$, $\wp$, sub *and* $\mathtt{sub}'$ *must be modified and extended in accordance with* PEL, PE$\wp$ *and* PEsub.

Alternative minimal sets of pairs of literals, alternative specifications of substitutions and alternative optimized prenexes are considered on alternative proof paths. The FOL-Decider considers only extensions of L, sub, and $\wp$ that are in accordance with PE. Any selection of pairs of literals and any specification of substitutions with respect to some optimized prenex is, roughly speaking, a fixed constituent in the search for a $\wedge$I-minimal proof along a proof path. On the one hand, this limits the search for proofs in the NNF-calculus to an effective search for $\wedge$I-minimal proofs. On the other hand, all combinatorial possibilities consisting of different alternatives for L, $\wp$ and sub are generated to ensure that a $\wedge$I-minimal proof of contradiction for D_i is found if D_i is refutable.

Remark 8.30. It is not necessary to reduce the search for a $\wedge$I-minimal proof to a search along proof paths on which $\mathtt{M_L P}$ and PE are applied. For a decision procedure based on the $\wedge$I-minimal proof strategy, it is sufficient to rely on a *finite* search space that *contains* the proof paths that are generated in accordance with $\mathtt{M_L P}$ and PE. In this sense, these principles are not necessary but rather are principles for an *effective* $\wedge$I-minimal proof search (cf. section 11).

The principles mentioned thus far may already constitute a sat-proof for a D_i. The following is a simplest possible example of such a proof.

Example 8.31. Let D_i be given as follows:

$$\exists y_1 F y_1 \wedge \exists y_2 G y_2 \wedge \forall x_1(\neg F x_1 \vee \neg G x_1) \tag{8.50}$$

There are only one optimized prenex $\wp$ and only one sub/L:

$$\wp: \quad \{y_1, y_2, x_1\} \tag{8.51}$$

$$\begin{aligned} \mathtt{sub/L}: \quad & \{\{\{\{\{x_1, y_1\}\}, \{\}, \{\}\}, \{F y_1, \neg F x_1\}\}, \\ & \{\{\{\{x_1, y_2\}\}, \{\}, \{\}\}, \{G y_2, \neg G x_1\}\}\} \end{aligned} \tag{8.52}$$

From the substitution specification sub given in (8.52), the following substitution list σ is obtained:

$$\sigma: \quad \{\{x_1, y_1, y_2\}\} \tag{8.53}$$

Thus, $\forall x_1(\neg Fx_1 \vee \neg Gx_1)$ in (8.50) must be multiplied to make it possible to replace x_{1_1} with y_1 and x_{1_2} with y_2:

$$\exists y_1 Fy_1 \wedge \exists y_2 Gy_2 \wedge \forall x_{1_1}(\neg Fx_{1_1} \vee \neg Gx_{1_1}) \wedge \forall x_{1_2}(\neg Fx_{1_2} \vee \neg Gx_{1_2}) \tag{8.54}$$

This results in the following modification of (8.52):

$$\begin{aligned} \mathtt{sub'/L'}: \quad & \{\{\{\{x_{1_1}, y_1\}\}, \{\}, \{\}\}, \{Fy_1, \neg Fx_{1_1}\}\}, \\ & \{\{\{\{x_{1_2}, y_2\}\}, \{\}, \{\}\}, \{Gy_2, \neg Gx_{1_2}\}\}\} \end{aligned} \tag{8.55}$$

According to PEL, all pairs of literals from (8.55) must be maintained when extending (8.55). To satisfy $\bot$P, (8.55) must be extended by either $\{Gy_2, \neg Gx_{1_1}\}$ or $\{Fy_1, \neg Fx_{1_2}\}$. However, both extensions violate $\mathtt{M_L P}$. If one were to select $\{Gy_2, \neg Gx_{1_1}\}$ in addition to $\{Fy_1, \neg Fx_{1_1}\}$, then $\{Gy_2, \neg Gx_{1_2}\}$ would be superfluous. Consequently, the $\wedge$I application for multiplying x_1 would be superfluous. This would violate the *Fundamental Principle* FP, because $\forall x_{1_2}(\neg Fx_{1_2} \vee \neg Gx_{1_2})$ would no longer be selected. A similar situation would result if one were to select $\{Fy_1, \neg Fx_{1_2}\}$ in addition to $\{Gy_2, \neg Gx_{1_2}\}$. Thus, the proof along the described proof path terminates because every extension violates some principle of the $\wedge$I-minimal proof strategy. Therefore, (8.50) is proven to be satisfiable (not refutable) since there are no alternatives to (8.51) and (8.52). No $\wedge$I-minimal proof path can be found. If (8.50) were refutable, however, a $\wedge$I-minimal proof would exist, according to Theorem 8.2.

9. Termination: Loop List

$\wedge$I multiplies an x variable μ. The multiplication of an x variable μ may necessitate additional multiplications of other x variables ν. A loop list codes sequences of such causally related multiplications of x variables.

To identify which multiplications of x variables ν are necessitated by the multiplication of an x variable μ, one must compare the list of substitutions prior to the last $\wedge$I multiplication ($\sigma 1$) with the substitution list after that $\wedge$I application ($\sigma 2$) on each proof path. After each application of $\wedge$I, the FOL-Decider generates new lists of the parameters sub/L and ρ. Alternatives of sub, L or ρ generate alternative proof paths. Alternative modifications may cause alternative substitution lists $\sigma 2$. For each proof path, the modification of the loop list on this path depends on the comparison between the list of substitutions prior to the last $\wedge$I multiplication ($\sigma 1$) with the substitution list on the actual path after that $\wedge$I application ($\sigma 2$). Since only variables that must be replaced with more than one y variable are considered, all partial lists of length 2 are deleted from $\sigma 1$ and $\sigma 2$. $\sigma 1$ can be generated from the sub prior to the $\wedge$I application[29], and $\sigma 2$ can be generated from the sub″ subsequent to the $\wedge$I application. Furthermore, it must be considered that L″ and sub″ contain the descendants of variables v bound by quantifiers from the multiplied expression $\forall \mu A(\mu)$. Let $\mathtt{vars_d}$ be the set of all descendants, and let $\mathtt{vars_a}$ be the set of all of their ancestors.

Definition 9.1. The multiplication of an x variable μ ($= \mathtt{xvar_c}$) *necessitates* the multiplication of an x variable ν ($= \mathtt{xvar_e}$) iff one of the following conditions is satisfied:

(1) ν is not a member of $\mathtt{vars_d}$, and
   (a) according to $\sigma 2$, ν is to be replaced with a y variable yvar that is not from $\mathtt{vars_d}$, and according to $\sigma 1$, ν is not to be replaced with yvar; or

[29] In fact, the FOL-Decider generates $\sigma 1$ from the first list of the heads of partial lists of the loop list prior to the $\wedge$I application.

(b) according to $\sigma 2$, ν is to be replaced with a y variable $\texttt{yvar}$ from $\texttt{vars}_\texttt{d}$, and according to $\sigma 1$, ν is not to be replaced with the ancestor of $\texttt{yvar}$; or
(c) according to $\sigma 2$, ν is to be replaced with two or more y variables from $\texttt{vars}_\texttt{d}$ that are descendants of one and the same y variable from $\texttt{vars}_\texttt{a}$.

(2) ν is a member of $\texttt{vars}_\texttt{d}$, and
(a) ν is a descendant of μ ($= \texttt{xvar}_\texttt{c}$) and occurs in $\sigma 2$; or
(b) according to $\sigma 2$, ν is to be replaced with a y variable $\texttt{yvar}$ that is not from $\texttt{vars}_\texttt{d}$, and according to $\sigma 1$, ν's ancestor is not to be replaced with $\texttt{yvar}$; or
(c) according to $\sigma 2$, ν is to be replaced with a y variable $\texttt{yvar}$ from $\texttt{vars}_\texttt{d}$, and according to $\sigma 1$, ν's ancestor is not to be replaced with the ancestor of $\texttt{yvar}$; or
(d) according to $\sigma 2$, ν must be replaced with two or more y variables from $\texttt{vars}_\texttt{d}$ that are descendants of one and the same y variable from $\texttt{vars}_\texttt{a}$.

(3) If the multiplication of an x variable μ necessitates the multiplication of an x variable ρ and the multiplication of ρ necessitates the multiplication of ν, then μ necessitates the multiplication of ν (transitivity).

In (2)(a), the multiplication of $\texttt{xvar}_\texttt{c}$ necessitates the multiplication of a descendant of the multiplied variable (cf. x_1 in Example 8.50 and x_3 in Example 9.19). In all other cases, the multiplication of $\texttt{xvar}_\texttt{c}$ necessitates that some other variable that is not a descendant of $\texttt{xvar}_\texttt{c}$ be replaced with a further y variable subsequent to the $\wedge$I application.

Definition 9.2. A *loop list* is a list of partial lists, each of which consists of a head and a body.

Definition 9.3. The *head* is a list that consists of

(1) a first list consisting of
(a) an x variable μ that must be replaced with more than one y variable according to σ[30] and
(b) the y variables $\texttt{yvars}$ that must replace μ according to σ and
(2) a second list $\texttt{sub/L}$ containing those pairs of literals from $\texttt{L}$ that contain literals in the scope of $\forall\mu$ in D_i^* plus the substitution specification $\texttt{sub}$ that specifies how the x variables of those pairs of literals are to be replaced with y variables.

Definition 9.4. The $\texttt{xvar}$-*head* is the head of the partial list with the x variable $\texttt{xvar}$ in the first position in the first list of the head. The $\texttt{xvar}$ *partial list* is the partial list with the x variable $\texttt{xvar}$ in the first position in the first list of the head.

Definition 9.5. $\texttt{xvar}_\texttt{c}$ is an x variable μ whose multiplication necessitates the multiplication of the x variable ν ($= \texttt{xvar}_\texttt{e}$) on a proof path.

Definition 9.6. The *body* of an $\texttt{xvar}_\texttt{e}$ partial list consists of all $\texttt{xvar}_\texttt{c}$-heads that necessitate the multiplication of $\texttt{xvar}_\texttt{e}$. In stage 1 on a proof path for D_i, the body is empty.

Subsequent to the application of $\wedge$I and the generation of $\texttt{L}''$, $\wp$ and $\texttt{sub}''$, the loop list is modified. To define the modification algorithm, different types of x variables must be distinguished (cf. Definitions 9.7 - 9.9).

Definition 9.7. A variable $\overline{\texttt{mxvar}}$ satisfies all of the following conditions:

[30]Prior to a $\wedge$I application, $\sigma = \sigma 1$; subsequent to the $\wedge$I application and the modification of the loop list, $\sigma = \sigma 2$.

(1) $\overline{\texttt{mxvar}}$ is required to be multiplied prior to the $\wedge$I application and therefore constitutes an xvar-head in the loop list prior to the $\wedge$I application.
(2) $\overline{\texttt{mxvar}}$ no longer needs to be multiplied subsequent to the $\wedge$I application according to $\sigma 2$.
(3) If $\overline{\texttt{mxvar}}$ is not identical to the multiplied x variable μ and $\overline{\texttt{mxvar}}$ is a member of $\texttt{vars}_\texttt{a}$, then no descendant of $\overline{\texttt{mxvar}}$ is required to be multiplied subsequent to the $\wedge$I application according to $\sigma 2$.

Definition 9.8. A variable newmxvar is an $\texttt{xvar}_\texttt{e}$ that must be multiplied due to the multiplication of μ ($= \texttt{xvar}_\texttt{c}$) in the last $\wedge$I application (cf. Definition 9.1).

Definition 9.9. A variable mxvar satisfies all of the following conditions:

(1) mxvar is required to be multiplied prior to the $\wedge$I application and therefore constitutes an xvar-head in the loop list prior to the $\wedge$I application.
(2) Either
  (a) mxvar is not a member of $\texttt{vars}_\texttt{a}$ and not a newmxvar but still is required to be multiplied subsequent to the $\wedge$I application according to $\sigma 2$, or
  (b) mxvar is a member of $\texttt{vars}_\texttt{a}$ and there are k ($k > 0$) descendants of mxvar, each of which is not a newmxvar but still is required to be multiplied subsequent to the $\wedge$I application according to $\sigma 2$.

Remark 9.10. It follows from Definition 9.7 and the $\wedge$I application that the multiplied x variable μ is a $\overline{\texttt{mxvar}}$. Its descendent might be a newmxvar but cannot be an mxvar.

The heads of the partial lists of the loop list are modified subsequent to the $\wedge$I application as dictated by the following algorithm.

Algorithm 9.11. Apply the following rules with regard to the different types of x variables:

Rule 1: Delete all $\overline{\texttt{mxvar}}$ partial lists.
Rule 2: Replace the mxvar partial lists as follows:
  (1) If mxvar is a member of $\texttt{vars}_\texttt{a}$, then replace the mxvar partial list with k partial lists. For each of the k ($k > 0$) descendants of mxvar as identified in Definition 9.9 (2)(b), modify the mxvar-head as follows:
    (a) Replace mxvar in the first list with the descendant of mxvar.
    (b) Replace the y variables yvars in the first list with the y variables $\texttt{yvars}'$ taken from $\sigma 2$.
    (c) Replace the second list sub/L with regard to $\texttt{sub}''/\texttt{L}''$.
  (2) If mxvar is not a member of $\texttt{xvars}_\texttt{a}$, then modify the mxvar-head of the mxvar partial list only in accordance with **Rule 2** (1)(b) and (c).
Rule 3: Add newmxvar partial lists to the loop list:
  (1) Take the μ partial list ($= \mu - PL$[31]) from the original loop list. Add the head of $\mu - PL$ to the body $B_{\mu old}$ of $\mu - PL$. The result is the body B_μ.
  (2) If
    (a) newmxvar is not a member of $\texttt{vars}_\texttt{d}$ and there is no newmxvar partial list in the loop list or
    (b) newmxvar is a member of $\texttt{vars}_\texttt{d}$ and either
      (i) there is no partial list for the ancestor of newmxvar in the loop list or

[31] I.e., the partial list of the multiplied x variable μ ($= \texttt{xvar}_\texttt{c}$).

(ii) `newmxvar` is a descendant of the multiplied x variable μ,

then add a `newmxvar` partial list $\texttt{newmxvar} - \texttt{PL}$ to the loop list. The head of $\texttt{newmxvar} - \texttt{PL}$ is a list with the following elements:

(a) a first list with `newmxvar` as the first element and a list of the y variables `yvars` with which `newmxvar` must be replaced according to $\sigma 2$ as the second element;

(b) a second list `sub`/`L` generated from `sub`$''$ and `L`$''$ with regard to the literals in the scope of the corresponding universally quantified expression in D_i^*.

The body of $\texttt{newmxvar} - \texttt{PL}$ is B_μ.

(3) If

(a) `newmxvar` is not a member of $\texttt{vars}_\texttt{d}$ and there is a `newmxvar` partial list $\texttt{newmxvar} - \texttt{PL}_\texttt{old}$ in the loop list or

(b) `newmxvar` is a member of $\texttt{vars}_\texttt{d}$ and there is a partial list $\texttt{newmxvar} - \texttt{PL}_\texttt{old}$ for the ancestor of `newmxvar` in the loop list,

then replace $\texttt{newmxvar} - \texttt{PL}_\texttt{old}$ with a partial list $\texttt{newmxvar} - \texttt{PL}$. The head of $\texttt{newmxvar} - \texttt{PL}$ is modified in accordance with **Rule 3** (2), and the body of $\texttt{newmxvar} - \texttt{PL}$ is the concatenation of the body of $\texttt{newmxvar} - \texttt{PL}_\texttt{old}$ with B_μ.

The body of a partial list codes the sequence of $\wedge$I multiplications that necessitate the multiplication of the x variable in the head of that partial list.

Remark 9.12. Rule 1 of Algorithm 9.11 implies that $\mu - PL$ is deleted. According to **Rule 3** (2), partial lists $\mu_i - PL$ for descendants of μ are added if μ_i is a `newmxvar`.

Remark 9.13. Prior to any $\wedge$I applications, the heads contain all x variables that must be multiplied due to the initial `sub` of the initial `L`, and the bodies of the partial lists in the loop list are empty. During the course of $\wedge$I applications, the lengths of the *bodies* will never decrease due to Algorithm 9.11. Instead, if a `newmxvar` exists, then the length of the body B_μ will increase compared with $B_{\mu old}$, and the length of the loop list may also increase. On the other hand, if there is no `newmxvar`, then the length of the *loop list* will decrease by at least 1 according to Algorithm 9.11. Since either there is a `newmxvar` or there is not, either the length of the *loop list* will decrease or the lengths of the *bodies* of the $\texttt{newmxvar} - \texttt{PL}$s in the loop list will increase over consecutive $\wedge$I applications along a proof path. Therefore, as long as `newmxvar`s result from $\wedge$I applications, the body lengths will increase.

The loop list makes it possible to specify definite termination conditions for the proof search in step 2 of the FOL-Decider.

Termination Criterion `T1` (*False Criterion*). *If the loop list = {}, then D_i is refutable.*

`T1` is the only criterion for refutation that is applied in step 2 of the FOL-Decider. If the loop list is empty, then it is no longer necessary to multiply any x variable. Therefore, an unambiguous substitution list σ, an optimal prenex $\wp$ and a minimal set `L` of unifiable pairs of literals have been found. Thus, all literals of D_i^* that are not contained in `L` can be deleted, and the resulting purged* anti-prenex normal form can be converted into a prenex normal form D_i^{**} with the optimal prenex $\wp$. Applying $\forall$E such that x variables are replaced with y variables in accordance with σ results in an explicit contradiction D_i^{***}. Consequently, the search for a proof of contradiction for D_i terminates with the result `False`.

During the course of $\wedge$I applications, a head can become part of the body of a partial list. The heads in the body of a partial list were once the heads of partial lists from previous loop lists on the proof path. To define the *Loop Criterion* for terminating proof paths that

do not result in a proof, let us define the concept of "isomorphic heads". Isomorphic heads can be mapped onto each other in a one-to-one manner.

Definition 9.14. Two heads are *isomorphic* iff they are identical apart from variable indices at levels > 1 and all variables that are identical up to level 1 can be mapped onto each other in a one-to-one manner such that the heads are mapped onto each other.

In a similar manner, I will also speak of *isomorphic universally quantified expressions* and *isomorphic $\wedge I$ applications*.

Whether two heads are isomorphic is decidable by (i) deciding whether the heads are identical up to level 1 according to a canonical order and, if so, (ii) replacing variables with indices at levels > 1 with variables with indices up to level 2 in a canonical way (such that different variables are replaced with different variables with indices of depth ≤ 2) and deciding whether the resulting heads are identical according to a canonical ordering. The FOL-Decider specifies an algorithm for this task. Since the details are trivial, it is sufficient to note that deciding on isomorphism reduces the maximum index level to 2.

Termination Criterion `T2` **(*Loop Criterion*).** *If the loop list contains a partial list PL with a body that contains a head that is isomorphic to the head of PL, then the proof search terminates on the corresponding proof path.*

Remark 9.15. As Example 8.31 illustrates, `T2` is not necessary for the termination of proof paths. `sat`-proofs of D_i may not even require the application of `T2`. `T2` is merely sufficient to terminate the search for a proof in the case that D_i is not refutable. In the absence of `T2`, the principles of the $\wedge$I-minimal search strategy are not sufficient to ensure the termination of the proof search in the case of an arbitrary, non-refutable D_i.

The *Loop Criterion* concerns the case in which the multiplication of an x variable necessitates an isomorphic $\wedge$I application, namely, one in which an isomorphic x variable must be replaced with isomorphic y variables (including y_0 with regard to $\wp$) in pairs of isomorphic literals that are to be similarly unified. This criterion represents the following situation: a $\wedge$I application is necessary to multiply a universally quantified expression $\forall\mu A(\mu)$ in order to replace the resulting descendants of μ with different y variables in different conjuncts, and this $\wedge$I application then necessitates an isomorphic application of $\wedge$I to an isomorphic universally quantified expression to unify similar pairs of literals in a similar manner.

I define the *correctness of the Loop Criterion* `T2` by means of the following theorem.

Theorem 9.16. *The* Loop Criterion `T2` *causes the termination only of proof paths that are not proof paths for $\wedge I$-minimal proofs.*

Proof. Every proof step on the proof path for a $\wedge$I-minimal proof is necessary (cf. Definition 8.1). A proof step of a $\wedge$I-minimal proof consists of one $\wedge$I application that is performed to multiply an x variable. The necessity of such a multiplication is computed with regard to `sub`, which specifies how to unify the pairs of literals in `L` with regard to $\wp$ (including the introduction of y_0 variables into `sub`). $\wedge$I applications might necessitate further $\wedge$I applications due to the resulting modifications and extensions of `L`, $\wp$ and, consequently, `sub`. A sequence of $\wedge$I applications that satisfies the *Loop Criterion* cannot be part of a $\wedge$I-minimal proof path because in this case, the result of this sequence of $\wedge$I applications is the condition for an isomorphic $\wedge$I application. The isomorphic multiplication of an isomorphic x variable for the sake of isomorphic substitutions of isomorphic pairs of literals

results in an isomorphic $\wedge$I application. Two isomorphic $\wedge$I applications cannot both be a necessary part of a $\wedge$I-minimal proof because the conditions for their application are the same in both cases. Thus, the proof search either enters a loop or proceeds with some alternative $\wedge$I applications that will avoid the loop in later proof steps. In the first case, the $\wedge$I applications will never terminate in a proof; in the second case, the sequence of $\wedge$I applications necessitating the isomorphic $\wedge$I application can be omitted if a proof is found. In either case, the sequence of $\wedge$I applications necessitating the isomorphic $\wedge$I application is not part of a $\wedge$I-minimal proof. Therefore, if a $\wedge$I-minimal proof for D_i exists, it cannot depend on a sequence of $\wedge$I applications that satisfies the *Loop Criterion*. A $\wedge$I-minimal proof can only be found on an alternative proof path. Thus, the search for a $\wedge$I-minimal proof on a proof path that satisfies the *Loop Criterion* can be terminated. □

Note that the *Prenex Condition* $\mathtt{P}_{\wp}$ is considered by introducing y_0 into `sub`. Therefore, isomorphic $\wedge$I applications also imply the similarity of $\wp$ with regard to the conditions for these isomorphic $\wedge$I applications.

Isomorphic $\wedge$I applications also imply similarity with regard to the number of different variables that are derivates of the same variable and, consequently, are taken from different conjuncts. The structural conditions for these isomorphic $\wedge$I applications are the same, and thus, one can conclude that causally related $\wedge$I applications cannot be part of a $\wedge$I-minimal proof.

`T1` and `T2` are the essential termination criteria.

Before we prove that these criteria suffice for terminating the $\wedge$I proof strategy, we need to consider the generation of multiplied expressions headed by universal quantifiers $\forall\mu_1$ and $\forall\mu_2$ that both originate from multiplying $\forall\mu$ but with non-isomorphic scopes. Suppose, for example, that an expression $\forall\mu A(\mu)$ is multiplied first such that the result is $\forall\mu_1 A(\mu_1) \wedge \forall\mu_2 A(\mu_2)$. Then, an expression $\forall\nu B(\nu)$ within the scope of $\forall\mu_1$ is multiplied (modifying $A(\mu_1)$ into $A'(\mu_1)$), which causes $\forall\mu_2 A(\mu_2)$ to need to be multiplied. In this situation, $\forall\mu_2 A(\mu_2)$ is no longer isomorphic to $\forall\mu_1 A'(\mu_1)$ because the scope $A'(\mu_1)$ contains one more multiplied expression than $A(\mu_2)$ does. Therefore, any multiplication of $\forall\mu_2$ that necessitates a further multiplication of $\forall\mu_1$ cannot necessitate an isomorphic $\wedge$I application of $\forall\mu_1$ because the numbers of literals in $A(\mu_2)$ and $A'(\mu_1)$ are no longer identical. However, the following lemma holds:

Lemma 9.17. *The process of successively multiplying expressions necessarily produces isomorphic multiplied expressions $\forall\rho_1 A(\rho_1)$ and $\forall\rho_2 A(\rho_2)$, where the scopes $A(\rho_1)$ and $A(\rho_2)$ each contain the same number of literals.*

Proof. Non-isomorphic multiplied expressions $\forall\mu_1 A'(\mu_1)$ and $\forall\mu_2 A(\mu_2)$ can occur only as a consequence of prior multiplications of quantifiers $\forall\nu$ within at least one of the scopes $A(\mu_1)$ and $A(\mu_2)$. However, the isomorphism of (i) multiplied universally quantified expressions of inner universally quantified expressions separated by existential quantifiers or conjunctions from $\forall\nu$ and (ii) outer universal quantifiers separated by existential quantifiers or conjunctions from $\forall\mu_1$ and $\forall\mu_2$ is not affected. Due to the finite length of D_i and the finite number of nested quantifiers within D_i, any sequence of successive multiplications of universally quantified expressions must, at some point, repeat the multiplication of such *inner* or *outer* universally quantified expressions. Therefore, any non-isomorphic sequence of successive multiplications of universally quantified expressions necessarily produces the repeated multiplication of isomorphic expressions with the same number of literals. □

Theorem 9.18. `T1` *and* `T2` *are sufficient to ensure that the search for a* $\wedge$*I-minimal proof of* D_i *in step 2 of the FOL-Decider terminates.*

Proof. In step 2 of the FOL-Decider, a search tree is generated in which each $\wedge$I application corresponds to a branching point. Subsequent to a $\wedge$I application, only a finite number of branches (= alternative proof paths) can be generated. This is so because the alternative proof paths depend only on differences in `L`, $\wp$, and `sub`, which, in turn, all depend on the finite length of D_i^* and are therefore finite. Due to the *Fundamental Principle* `FP`, every conjunct is maintained once selected, and thus, the number of selected conjuncts increases with each $\wedge$I application. Therefore, new branches at the end of the search tree differ from previous branching points on the same proof path by their increased number of selected conjuncts, and the search tree terminates if it terminates in depth.

That the search tree terminates in depth is ensured by the criteria `T1` and `T2`, which are related to the loop list. Any loop list is finite because only a finite number of x variables can be replaced with only a finite number of y variables in only a finite number of pairs of literals due to the finite length of D_i^*. If the initial loop list prior to any $\wedge$I application is empty, then the search for a proof terminates due to `T1` in stage 1. If the multiplication of an x variable μ does not necessitate any further multiplication of an x variable, then the length of the loop list decreases by at least 1 because at least the μ partial list is deleted (cf. Remark 9.12).

As explained in Remark 9.13, subsequent to a $\wedge$I application, either the length of the loop list decreases by 1 or an increase in body length occurs. Thus, as long as the loop list is not empty, and `T1` therefore does not apply, the lengths of the bodies (= numbers of heads in the bodies) in the loop list increase over consecutive $\wedge$I applications. Consequently, the condition of the *Loop Criterion* must be satisfied after a finite number of steps given Lemma 9.17. This is so because the number of non-isomorphic heads with the same types and numbers of literals from isomorphic scopes is finite due to the fact that the identification of isomorphic heads reduces the index depth to ≤ 2. Therefore, since the process of successive multiplication of universally quantified expressions necessarily results in the multiplication of isomorphic expressions (Lemma 9.17), the substitutions of pairs of literals containing the literals from the scope of the multiplied universal quantifier will, at some stage, be isomorphic. At this point, the $\wedge$I application will be identified by the *Loop Criterion* as a superfluous repetition in the proof search.

The *Loop Criterion* must be satisfied after a finite number of steps even in the case that the y variables `yvars` contain several derivates of the same y variable. This is true because the number of possible substitutions of an x variable μ is bounded by its occurrences in positions in different literals in the (finite) scope $A(\mu)$ in $\forall\mu A(\mu)$. Isomorphic heads contain identical numbers of these positions; thus, there is only a finite number of possible isomorphic heads due to the finite length of isomorphic multiplications of $\forall\mu A(\mu)$.

Therefore, some head in the body of some partial list will become isomorphic to the head of that partial list during the course of $\wedge$I applications unless the loop list is emptied. □

Example 9.19. The following formula (9.1) is a simple example of a formula that has only infinite models. However, there is no need to refer to model theory to prove its non-refutability (satisfiability). It can be proven to be non-refutable on the basis of the $\wedge$I-minimal proof strategy and the *Loop Criterion*.

Let D_i be given as follows:

$$\forall x_1 \exists y_1 (Fx_1y_1 \wedge \forall x_3 (Fx_3y_1 \vee \neg Fx_3x_1)) \wedge \forall x_2 \neg Fx_2x_2 \tag{9.1}$$

(9.1) is equivalent to a formula presented by [Boerger et al. (2001)], p. 33, as an example of a formula that has only infinite models. It is not decided in step 1 of the FOL-Decider. Step 2 starts with the generation of `L`, $\wp$, and `sub` prior to any ∧I application. There is only one initial minimal set of pairs of literals:[32]

$$\texttt{L}: \quad \{\{Fx_1y_1, \neg Fx_3x_1\}, \{Fx_3y_1, \neg Fx_2x_2\}\} \tag{9.2}$$

There is also only one optimized prenex (y_0 precedes any prenex by definition):

$$\wp: \quad \{y_0, x_1, y_1, x_2, x_3\} \tag{9.3}$$

In addition, there is only one `sub` that specifies how to unify (9.2) in relation to (9.3). x_2 and x_3 must both be replaced with y_1 in the second pair of literals $\{Fx_3y_1, \neg Fx_2x_2\}$. x_1 must be replaced with y_1 in the second position in $\neg Fx_3x_1$ to unify the first pair of literals $\{Fx_1y_1, \neg Fx_3x_1\}$. x_1 and x_3 must both be replaced with the same y variable in the first positions in the literals. Thus, σ is unambiguous as long as $\texttt{P}_\wp$, and consequently (9.3), is not considered. However, condition $\texttt{P}_\wp$ on p. 48 is satisfied only if x_1 is replaced with y_0 in addition to y_1 since y_1 is to the right of x_1 in (9.3). Thus, x_1 must be multiplied to replace x_{1_2} with y_{1_1} subsequent to the ∧I application. In addition to the mentioned occurrence of x_1 in the second position in $\neg Fx_3x_1$, x_1 occurs only in the first position in Fx_1y_1 in the first pair of literals $\{Fx_1y_1, \neg Fx_3x_1\}$ in (9.2). At this position, x_1 must be replaced with y_0 because a pair of literals in which both literals contain an x variable `xvar` is unifiable only if in at least one of the two literals, `xvar` must be replaced with a y variable that is to the left of `xvar` in $\wp$.[33] As a consequence of replacing x_1 in the first position in Fx_1y_1 with y_0, x_3 must also be replaced with y_0 in the first position in $\neg Fx_3x_1$. Thus, the following `sub`/`L` is obtained:

$$\{\{\{\{\{x_1, y_1\}\}, \{\{x_1, x_3, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_3x_1\}\}, \tag{9.4}$$
$$\{\{\{\{x_2, y_1\}, \{x_3, y_1\}\}, \{\}, \{\}\}, \{Fx_3y_1, \neg Fx_2x_2\}\}\} \tag{9.5}$$

This results in the following substitution list σ:

$$\{\{x_1, y_0, y_1\}, \{x_2, y_1\}, \{x_3, y_0, y_1\}\} \tag{9.6}$$

Since x_1 as well as x_3 must be replaced with y_0 and y_1, the initial loop list contains two partial lists, (9.7) and (9.8). Since this loop list is the initial one, the bodies of the partial lists are empty. I highlight the first lists in boldface:

$$\{\{\{\mathbf{x_1, y_0, y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_3, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_3x_1\}\},$$
$$\{\{\{\{x_2, y_1\}, \{x_3, y_1\}\}, \{\}, \{\}\}, \{Fx_3y_1, \neg Fx_2, x_2\}\}\}\}, \{\}\}, \tag{9.7}$$
$$\{\{\{\mathbf{x_3, y_0, y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_3, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_3, x_1\}\},$$
$$\{\{\{\{x_2, y_1\}, \{x_3, y_1\}\}, \{\}, \{\}\}, \{Fx_3, y_1, \neg Fx_2, x_2\}\}\}\}, \{\}\}\} \tag{9.8}$$

[32] Fx_1y_1 and $\neg Fx_2x_2$ do not constitute a unifiable pair of literals since $\forall x_1 \exists y_1 Fx_1y_1 \wedge \forall x_2 \neg Fx_2x_2$ is not refutable.

[33] This condition is similar to C3U1 from [Lampert (2019b)], p. 28, and is implemented in step 2 of the FOL-Decider.

Since x_1 as well as x_3 must be replaced with more than one y variable, $\wedge$I must be applied to multiply x_1 and x_3. Both are necessary to unify all pairs of literals from (9.2). One could consider the multiplication of different universally quantified expressions in different orders on different proof paths. However, this would be ineffective. Because the x variables from lists of lengths > 2 in σ must be multiplied anyway, one of the x variables to be multiplied can be selected arbitrarily. If a $\wedge$I-minimal proof exists, it will be found with any order of multiplication of the universally quantified expressions. Thus, the following principle can be established:

Principle of Commutativity (PC). *If several x variables must be multiplied, any of them can be multiplied first.*

However, this does not mean that the number of $\wedge$I applications on a proof path and the number of alternative proof paths do not depend on the order in which universally quantified expressions are multiplied. Not the correctness but the efficiency of the $\wedge$I-minimal proof strategy depends on which universally quantified expression is selected to be multiplied first. Therefore, the proof search depends on the implemented strategy for ordering the necessary $\wedge$I applications. In most cases, the FOL-Decider multiplies the outermost universally quantified expressions before the innermost ones because inner expressions are also multiplied in the process of multiplying outer ones. However, the multiplication of existentially quantified expressions increases the complexity of the proof search. For this reason, universally quantified expressions that do not contain existentially quantified expressions are multiplied first. Consequently, $\forall x_3(Fx_3y_1 \vee \neg Fx_3x_1)$ is multiplied first in (9.1). This results in the following D_i^*:

$$\begin{aligned}&\forall x_1 \exists y_1(Fx_1y_1 \wedge \forall x_{3_1}(Fx_{3_1}y_1 \vee \neg Fx_{3_1}x_1) \wedge \\ &\forall x_{3_2}(Fx_{3_2}y_1 \vee \neg Fx_{3_2}x_1)) \wedge \forall x_2 \neg Fx_2x_2 \end{aligned} \tag{9.9}$$

The resulting $\mathtt{sub}'/\mathtt{L}'$ is as follows:

$$\{\{\{\{\{x_1, y_1\}\}, \{\{x_1, x_{3_1}, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_{3_1}x_1\}\}, \tag{9.10}$$

$$\{\{\{\{x_2, y_1\}, \{x_{3_2}, y_1\}\}, \{\}, \{\}\}, \{Fx_{3_2}y_1, \neg Fx_2x_2\}\}\} \tag{9.11}$$

Finally, the resulting extended optimized prenex $\wp'$ is

$$\wp' : \quad \{y_0, x_1, y_1, x_2, x_{3_1}, x_{3_2}\} \tag{9.12}$$

Since the multiplied universally quantified expression contains $\vee$, $\mathtt{L}'$ must be extended. There is only one possible extension that satisfies $\perp\mathtt{P}$, $\mathtt{M_L P}$ and PEL, namely, the addition of the unifiable pair of literals $\{Fx_{3_1}y_1, \neg Fx_{3_2}x_1\}$. Since (9.10) and (9.11) must be maintained, according to PE, there are exactly two alternatives for specifying the substitutions of this pair of literals:

$$\textit{Alternative}\,1: \quad \{\{\{\{x_1, y_1\}\}, \{\{x_{3_1}, x_{3_2}, y_1\}\}, \{\}\}, \{Fx_{3_1}y_1, \neg Fx_{3_2}x_1\}\} \tag{9.13}$$

$$\textit{Alternative}\,2: \quad \{\{\{\{x_1, y_1\}\}, \{\{x_{3_1}, x_{3_2}, y_0\}\}, \{\}\}, \{Fx_{3_1}y_1, \neg Fx_{3_2}x_1\}\} \tag{9.14}$$

Alternative 1 is realized on proof path 1, and *Alternative 2* is realized on an alternative proof path 2. In *Alternative 1*, x_{3_1} must be replaced with y_1 in addition to y_0 in (9.10). In *Alternative 2*, x_{3_2} must be replaced with y_0 in addition to y_1 in (9.11). If one counts not

merely the ∧I applications on a single proof path but the absolute number of ∧I applications, then the multiplication of x_{3_1} is the second ∧I application, and the multiplication of x_{3_2} is the third. In both cases, applying ∧I results in two further proof paths. In the following, I consider *Alternative 1* only on proof path 1; proof path 2 terminates similarly within the same number of steps.

The modified loop list for *Alternative 1* contains two partial lists, (9.15) and (9.16), since x_1 and x_{3_1} must be multiplied. I highlight the first lists in the head and in the body of the partial lists in boldface:

$$\begin{gathered}\{\{\{\{\mathbf{x_1}, \mathbf{y_0}, \mathbf{y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_{3_1}, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_{3_1}x_1\}\}, \\ \{\{\{\{x_1, y_1\}\}, \{\{x_{3_1}, x_{3_2}, y_1\}\}, \{\}\}, \{Fx_{3_1}y_1, \neg Fx_{3_2}x_1\}\}, \\ \{\{\{\{x_2, y_1\}, \{x_{3_2}, y_1\}\}, \{\}, \{\}\}, \{Fx_{3_2}y_1, \neg Fx_2, x_2\}\}\}\}, \{\}\},\end{gathered} \tag{9.15}$$

$$\begin{gathered}\{\{\{\mathbf{x_{3_1}}, \mathbf{y_0}, \mathbf{y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_{3_1}, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_{3_1}x_1\}\}, \\ \{\{\{\{x_1, y_1\}\}, \{\{x_{3_1}, x_{3_2}, y_1\}\}, \{\}\}, \{Fx_{3_1}y_1, \neg Fx_{3_2}x_1\}\}\}\},\end{gathered} \tag{9.16}$$

$$\begin{gathered}\{\{\{\mathbf{x_3}, \mathbf{y_0}, \mathbf{y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_3, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_3x_1\}\}, \\ \{\{\{\{x_2, y_1\}, \{x_3, y_1\}\}, \{\}, \{\}\}, \{Fx_3y_1, \neg Fx_2x_2\}\}\}\}\}\end{gathered}$$

The body of partial list (9.15) is still empty. In the case of partial list (9.16), the *Loop Criterion* is not yet satisfied because the second pairs of literals in the head and the list in the body are not isomorphic.

Although x_1 is still to be replaced with y_0 and y_1, derivates of x_3 are to be multiplied prior to (derivates of) x_1, according to the *Commutative Principle* PC (cf. p. 64). Multiplying x_{3_1} results in the following D_i^*:

$$\begin{gathered}\forall x_1 \exists y_1 (Fx_1y_1 \wedge \\ \forall x_{3_{1_1}} (Fx_{3_{1_1}}y_1 \vee \neg Fx_{3_{1_1}}x_1) \ \wedge \forall x_{3_{1_2}} (Fx_{3_{1_2}}y_1 \vee \neg Fx_{3_1}x_1) \wedge \\ \forall x_{3_2} (Fx_{3_2}y_1 \vee \neg Fx_{3_2}x_1)) \wedge \forall x_2 \neg Fx_2x_2\end{gathered} \tag{9.17}$$

The resulting $\mathtt{sub}'/\mathtt{L}'$ is as follows:

$$\{\{\{\{x_1, y_1\}\}, \{\{x_1, x_{3_{1_1}}, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_{3_{1_1}}x_1\}\}, \tag{9.18}$$

$$\{\{\{\{x_2, y_1\}, \{x_{3_2}, y_1\}\}, \{\}, \{\}\}, \{Fx_{3_2}y_1, \neg Fx_2x_2\}\}\}, \tag{9.19}$$

$$\{\{\{\{x_1, y_1\}\}, \{\{x_{3_{1_2}}, x_{3_2}, y_1\}\}, \{\}\}, \{Fx_{3_{1_2}}y_1, \neg Fx_{3_2}x_1\}\}\}\} \tag{9.20}$$

Finally, the resulting extended optimized prenex $\wp'$ is

$$\wp' : \quad \{y_0, x_1, y_1, x_2, x_{3_{1_1}}, x_{3_{1_2}}, x_{3_2}\} \tag{9.21}$$

Again, $\mathtt{L}'$ must be extended, and there is only one possible extension that satisfies ⊥P, $\mathtt{M_L P}$ and PEL, namely, the addition of the unifiable pair of literals $\{Fx_{3_{1_1}}y_1, \neg Fx_{3_{1_2}}x_1\}$. As before, there are two and only two alternatives for specifying the substitutions of this pair of literals:

$$\textit{Alternative}\,1' : \quad \{\{\{\{x_1, y_1\}\}, \{\{x_{3_{1_1}}, x_{3_{1_2}}, y_1\}\}, \{\}\}, \{Fx_{3_{1_1}}y_1, \neg Fx_{3_{1_2}}x_1\}\} \tag{9.22}$$

$$\textit{Alternative}\,2' : \quad \{\{\{\{x_1, y_1\}\}, \{\{x_{3_{1_1}}x_{3_{1_2}}, y_0\}\}, \{\}\}, \{Fx_{3_{1_1}}y_1, \neg Fx_{3_{1_2}}x_1\}\} \tag{9.23}$$

In the case of *Alternative 1'*, $x_{3_{1_1}}$ must again be replaced with y_0 and y_1 to unify the pairs of literals in (9.18) and (9.22), which are isomorphic to (9.10) and (9.13). Therefore, the loop list satisfies the *Loop Criterion* and terminates the proof path that extends (9.18) - (9.20) by adding (9.22). Since the x_1 partial list is irrelevant for the application of the *Loop Criterion*, I quote only the $x_{3_{1_1}}$ partial list:

$$\begin{array}{c}\{\{\{\mathbf{x_{3_{1_1}}}, \mathbf{y_0}, \mathbf{y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_{3_{1_1}}, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_{3_{1_1}}x_1\}\},\\ \{\{\{x_1, y_1\}\}, \{\{x_{3_{1_1}}, x_{3_2}, y_1\}\}, \{\}\}, \{Fx_{3_{1_1}}y_1, \neg Fx_{3_2}x_1\}\}\}\},\\ \{\{\{\mathbf{x_3}, \mathbf{y_0}, \mathbf{y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_3, y_0\}\}, \{\}\}, \{\{Fx_1y_1, \neg Fx_3x_1\}\}, \qquad (9.24)\\ \{\{\{x_2, y_1\}, \{x_3, y_1\}\}, \{\}, \{\}\}, \{Fx_3y_1, \neg Fx_2x_2\}\}\}\},\\ \{\{\mathbf{x_{3_1}}, \mathbf{y_0}, \mathbf{y_1}\}, \{\{\{\{x_1, y_1\}\}, \{\{x_1, x_{3_1}, y_0\}\}, \{\}\}, \{Fx_1y_1, \neg Fx_{3_1}x_1\}\},\\ \{\{\{x_1, y_1\}\}, \{\{x_{3_1}, x_{3_2}, y_1\}\}, \{\}\}, \{Fx_{3_1}y_1, \neg Fx_{3_2}x_1\}\}\}\}\end{array}$$

The last list in the body is isomorphic to the head of (9.24).

The resulting sub/L on the proof path that consists of (9.18) - (9.20) and (9.23) (= *Alternative 2'*) is not eliminated by the *Loop Criterion* T2. This is so because the pairs of literals in (9.20) and (9.23) that contain literals in the scope of $\forall x_{3_{1_2}}$ are not isomorphic to the pairs of literals in (9.10) and (9.13) that contain literals selected from the scope of $\forall x_{3_1}$ in the previous sub/L resulting from the first $\wedge$I application.

Regarding the sub/L resulting from *Alternative 2'*, $x_{3_{1_2}}$ must be replaced with y_0 and y_1. The multiplication of $x_{3_{1_2}}$ (= the fourth $\wedge$I application in the proof search) results in two alternative sets of $\mathtt{sub}''/\mathtt{L}''/\wp'$. The modification of the loop list in accordance with these $\mathtt{sub}''/\mathtt{L}''/\wp$ alternatives satisfies the conditions of the *Loop Criterion* in both cases. In one case, $x_{3_{1_{2_1}}}$ must be replaced with y_0 and y_1; in the other, $x_{3_{1_{2_2}}}$ must be replaced with y_0 and y_1; and in both cases, the pairs of literals and their substitutions are isomorphic to those of (9.20) and (9.23) (*Alternative 2'*).

The multiplication of x_{3_2} (= the third $\wedge$I application in the proof search) yields results similar to those of the multiplication of x_{3_1}; the multiplication of $x_{3_{2_1}}$ (= the fifth $\wedge$I application in the proof search) subsequent to the multiplication of x_{3_2} yields results similar to those of the multiplication of $x_{3_{1_2}}$. Thus, the proof search terminates after 5 steps due to T2. Figure 2 depicts the relation between the number of proof steps or $\wedge$I applications (x axis) and the number of proof paths (y axis). The number of proof paths can be reduced by only 1 in one proof step. In a typical sat-proof, the number of proof paths initially increases rapidly and then, at a certain point, begins to slowly decrease. By contrast, False-proofs terminate immediately at any step and any number of proof paths.

Remark 9.20. The D_i given in (9.25) is equivalent to a formula from [Dreben (1979)], p. 120, which is another formula that has only infinite models:

$$\forall x_1 \exists y_1 (\forall x_4 (\neg P x_1 x_4 \vee Q y_1 x_4) \wedge \forall x_5 (\neg Q x_1 x_5 \vee Q y_1 x_5)) \wedge \forall x_2 \neg Q x_2 x_2 \wedge \forall x_3 P x_3 x_3 \quad (9.25)$$

Like (9.1), (9.25) can be rather rapidly proven by the FOL-Decider to be non-refutable (cf. figure 3).

(9.1) (= SYO637+1.p in the TPTP library) and (9.25) (= SYO635+1.p in the TPTP library) are examples of formulas with only infinite models for which the FOL-Decider

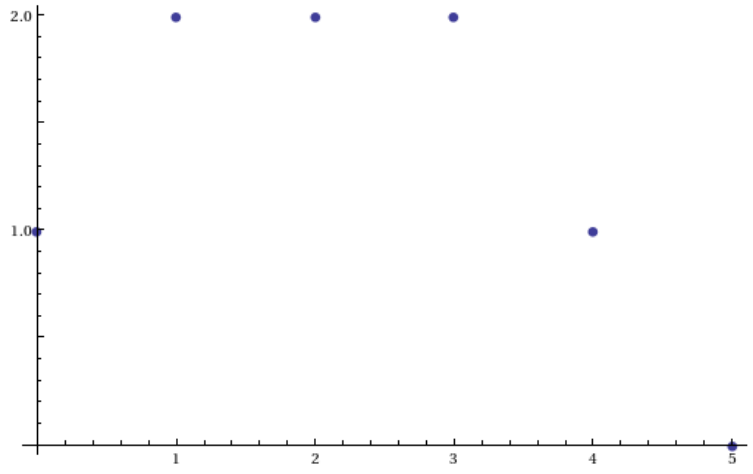

Figure 2: Number of proof paths vs. number of proof steps in the `sat`-proof of (9.1)

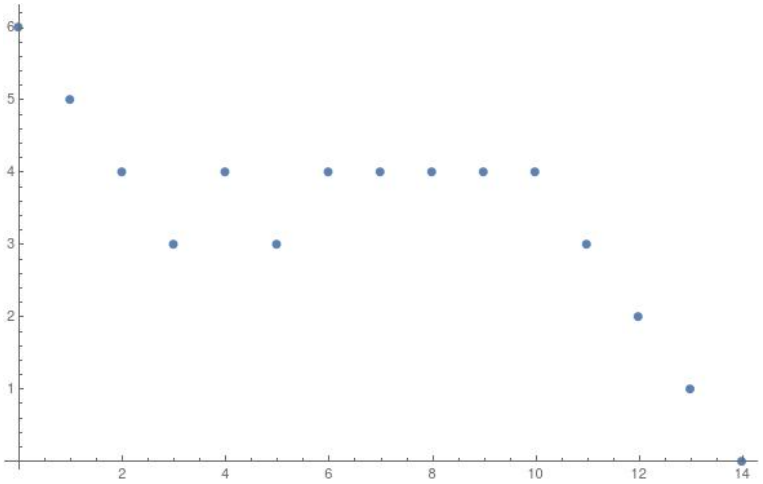

Figure 3: Number of proof paths vs. number of proof steps in the `sat`-proof of (9.25)

reaches a decision, whereas these formulas are not decided by other logic engines, such as Beagle, Darwin, i-Prover, E, Vampire and SPASS.

10. Output

If the FOL-Decider finds a ∧I-minimal proof for D_i in step 2, then the output, in addition to D_i, is as follows:

(1) the rectified and purged* anti-prenex normal form D_i^*,
(2) the minimal set `L` of unifiable pairs of literals,
(3) the unambiguous substitution list σ that specifies how the x variables must be replaced with y variables such that all pairs of literals from `L` are unified, and
(4) an optimal prenex $\wp$ that allows the substitutions in σ to be performed by applying ∀E.

Thus, a recipe for a proof of contradiction is provided. From D_i^*, one can read off which universally quantified expressions in D_i must be multiplied and which conjuncts can be deleted; $\wp$ identifies the prenex normal form D_i^{**} that is to be generated from D_i^* by applying PN laws; σ determines how $\forall$E must be applied to derive an explicit contradiction D_i^{***} from D_i^{**}; and `L` specifies the pairs of literals that must be unified such that each disjunct of the DNF matrix of D_i^{***} contains an explicit contradiction. For an example, cf. Example 8.28, p. 52, with D_i = (8.42), D_i^* = (8.46), `L` = (8.47), σ = (8.48), and $\wp$ = (8.49).

The FOL-Decider ultimately returns `False` if all disjuncts D_i of the FOLDNF resulting from step 1 of the FOL-Decider are proven to be refutable. In this case, the user receives the FOLDNF that is sat-equivalent to the input formula ϕ and all recipes for the $\wedge$I-minimal proofs for the disjuncts D_i of the FOLDNF.

Once the proof search for a D_i terminates without a $\wedge$I-minimal proof having been found, the FOL-Decider returns `sat`. In this case, D_i and, therefore, the FOLDNF and the input formula ϕ are proven to be non-refutable. The user can ask for details of the `sat`-proofs, e.g., diagrams such as figures 2 and 3; the D_i^*, `sub`/`L`, and $\wp$ for each proof step; or the applications of the various principles of the $\wedge$I-minimal proof strategy.

11. Effective Proof Search

The search for $\wedge$I-minimal proofs can be optimized from several perspectives. Three optimization strategies are already implemented as optional tools. First, one can minimize the FOLDNFs in step 1 of the FOL-Decider using the algorithm described in [Lampert (2017a)]. Furthermore, the scopes of existential quantifiers can be further minimized in step 1 of the FOL-Decider through $\exists$M optimization (cf. section 7 of [Lampert (2019b)] and footnote 19). However, both of these tools sometimes speed up the evaluation but sometimes slow it down. Finally, one can also apply an optional model-theoretic tool to search for a model up to an upper bound independent of the proof-theoretic strategy of the FOL-Decider. In addition, several optional tools can be applied to provide models in the case of a `sat`-proof. However, the decision procedure based on the $\wedge$I-minimal strategy is independent of model theory. If one intends to understand this strategy, one should abstain from invoking model theory.

Three further optimization strategies have not yet been implemented. First, the combinatorial generation of `sub`/`L`/$\wp$ alternatives and the *Loop Criterion* may be restricted by further criteria. Second, the FOL-Decider does not currently apply any principles that rely on comparing alternative proof paths, apart from deleting duplicates. Third, one could classify patterns of `sub`/`L`/$\wp$ for the termination of proof paths. Furthermore, many improvements to both the concrete implementation and the proof search strategy are possible. However, the FOL-Decider is not designed to implement a fast and effective proof search. In this respect, other logic engines are preferable. Instead, the intent in developing the FOL-Decider was to implement the $\wedge$I-minimal proof strategy to demonstrate its feasibility.

If one is looking for a transparent and elegant proof search that implements the logical ideas underlying the $\wedge$I-minimal strategy, then principles and criteria that allow one to terminate proof paths as soon as possible are indispensable. Any optimization, however, is prone to error. For this reason, I refer to a "maximal search tree" instead of an "effective search tree" in the following.

Definition 11.1. A *maximal search tree* generates all combinatorially possible non-duplicate alternatives of the following:

(1) D_i^* resulting from different selections of the order in which to multiply universally quantified expressions,
(2) minimal and non-minimal sets $\mathtt{L}^*$ of unifiable pairs of literals,
(3) optimized and not optimized (logically valid) prenexes $\wp^*$, and
(4) all alternative specifications $\mathtt{sub}^*$ of substitutions of x variables in xx positions with y variables for unification

such that the principles FP, $\perp$P, and $\mathtt{P}_\wp$ and the termination criteria T1 and T2 are satisfied.

Remark 11.2. I abstain here from providing an exact algorithmic definition of "all combinatorial alternatives of D_i^*, $\mathtt{L}^*$, $\wp^*$, and $\mathtt{sub}^*$ in the course of each $\wedge$I application". Likewise, I abstain from specifying formulas for calculating the number of these alternatives. It is sufficient to recognize that they are finite due to the finite numbers of literals, variables and universal quantifiers in each step on each proof path. The algorithmic specification of FP, $\perp$P, and $\mathtt{P}_\wp$ for a $\wedge$I-minimal proof search is also trivial. Thus, it should be accepted that a maximal search tree for D_i can be generated.

Remark 11.3. A maximal search tree contains all proof paths that satisfy the principles $\mathtt{M_L}$P, PE, and PC, but it is not restricted to these proof paths. FP, $\mathtt{P}_\wp$, and $\perp$P are *necessary principles* for a $\wedge$I-minimal proof search, and T1 and T2 are *necessary* criteria for the termination of a proof search. By contrast, $\mathtt{M_L}$P, PE, and PC are principles of an *effective* search strategy for $\wedge$I-minimal proofs.

Remark 11.4. The proof of Theorem 9.18 does not depend on the efficiency of a $\wedge$I-minimal proof search. As long as the loop list is not empty, the body lengths increase with every $\wedge$I application that does not decrease the length of the loop list. Consequently, the head of some partial list will become isomorphic to some head in the body of that partial list within a finite number of steps because only a finite number of isomorphic heads can be generated from D_i.

Theorem 11.5. *A maximal search tree for a D_i is finite.*

Proof. The proof of this theorem follows from Theorem 9.18 and Remark 11.4 and can be summarized as follows:

(1) The combinatorial alternatives of D_i^*, $\mathtt{L}^*$, $\wp^*$ and $\mathtt{sub}^*$ in the course of each $\wedge$I application are finite. Therefore, each branching point is finite.
(2) If the search for a proof along a proof path does not terminate due to an empty loop list or the necessary principles for the termination of a $\wedge$I-minimal proof search, then the proof search will eventually lead to an isomorphic $\wedge$I application due to the finite length of D_i. Therefore, T2 applies unless a proof path is terminated due to the necessary principles T1 or FP, $\perp$P and $\mathtt{P}_\wp$.

□

12. Correctness

Proving the correctness of the FOL-Decider requires proving that the FOL-Decider returns False iff the input formula ϕ is refutable. The proof in the direction from left to right is easy.

Theorem 12.1. *If the FOL-Decider returns* `False`*, then the input formula* ϕ *is refutable.*

Proof. The result of step 1 of the FOL-Decider is an FOLDNF. According to Theorem 5.8, the input formula ϕ is sat-equivalent to its FOLDNF. If the FOL-Decider returns `False` as a result in step 1, then the simple False/sat check applied in step 1 and referred to in the proof of Theorem 5.8 already identifies the FOLDNF and, therefore, the sat-equivalent initial formula ϕ as refutable. Otherwise, the FOL-Decider returns `False` iff it returns `False` for each disjunct D_i from disjuncts D_1 to D_n of the FOLDNF. Since ϕ is refutable iff each disjunct D_i is refutable, ϕ is refutable if each D_i is indeed refutable in the case that the FOL-Decider returns `False` for each D_i as a result of step 2. If the FOL-Decider returns `False` for D_i as a result of step 2, it returns a recipe for a proof of contradiction for D_i within the NNF-calculus. This recipe determines the necessary applications of $\wedge$I, an optimal prenex that can be generated via PN laws and the applications of $\forall$E that are needed to derive an explicit contradiction. From the correctness of the NNF-calculus (cf. Theorem 3.3 in [Lampert (2019b)]), it follows that each D_i and, therefore, the FOLDNF and ϕ are indeed refutable. □

For a complete and detailed proof in the direction from right to left, it is necessary to replace the definitions and principles serving as the basis of the $\wedge$I-minimal proof search strategy presented in this paper with their exact algorithmic implementations. In particular, this concerns the computations of `L`, $\wp$, and `sub` and their modification and extensions. However, I abstain from this here. It is not necessary to describe the algorithmic details if what is in question is not the correctness of the specific program (the FOL-Decider) but rather the correctness of its underlying decision strategy. To prove the decidability of D_i, it is sufficient to prove that the finite *maximal* search tree for $\wedge$I-minimal proofs in the NNF-calculus contains a $\wedge$I-minimal proof if D_i is refutable.

Definition 12.2. A maximal search tree *contains* a $\wedge$I-minimal proof iff it generates a combination of D_i^*, `L`, $\wp$, and `sub` that constitutes a $\wedge$I-minimal proof.

Lemma 12.3. *A maximal search tree contains all* $\wedge$*I-minimal proofs.*

Proof. According to Definition 11.1, a maximal search tree generates all combinatorially possible D_i^*, $\texttt{L}^*$, $\wp^*$, and $\texttt{sub}^*$. By definition, this totality of combinations contains all combinations of D_i^*, `L`, $\wp$, and `sub` that constitute $\wedge$I-minimal proofs (cf. Remark 11.3). □

Referring to a maximal proof tree circumvents the necessity of proving the correctness of the principles $\texttt{M}_\texttt{L}\texttt{P}$, `PE`, and `PC` of an effective search strategy for $\wedge$I-minimal proofs. The decidability of FOL through the $\wedge$I-minimal proof strategy in the NNF-calculus can be proven on the basis of Lemma 12.3 without considering the details of its implementation in the FOL-Decider.

Theorem 12.4. FOL *is decidable.*

Proof. According to Theorem 5.8, ϕ is sat-equivalent to its FOLDNF. Thus, ϕ is decidable if D_i is decidable. That D_i is decidable follows from Theorem 11.5 (finiteness of the maximal search tree), Lemma 12.3 (completeness of a maximal search tree with respect to $\wedge$I-minimal proofs), Theorem 8.2 (completeness of $\wedge$I-minimal proofs of D_i in the complete NNF-calculus), Theorem 12.1 (correctness of `False` outputs) and Theorem 9.16 (correctness of the *Loop Criterion* for the ultimate termination of proof paths). □

Remark 12.5. Theorem 12.4 contradicts the meta-mathematical proof of the Church-Turing theorem. In contrast to this meta-mathematical proof, the proof of FOL's decidability rests on nothing but purely logical considerations concerning $\wedge$I-minimal proofs. This proof is independent of the meta-mathematical proof method, which relies on translations of recursive or Turing-computable functions into the language of logic in order to express those computable functions. Given the correctness of the purely logical reasoning concerning the decidability of FOL, this suggests that the translation procedures are not correct in the case of the decision function for FOL-provability.

Acknowledgements. I am grateful to Anderson Nakano for discussing my algorithm and its proof as well as the critique of the Church-Turing theorem. Furthermore, I am grateful to Karsten Müller, Michael Taktikos, Pietro Fornara, Yvonne Lampert, Markus Säbel, Stefan Steins, Geoff Suttcliffe and Victor Rodych.

References

[Boerger et al. (2001)] Börger, E., Grädel, E. & Gurevich, Y.: *The Classical Decision Problem*, Springer.

[Boolos et al. (2003)] Boolos, G.S., Burgess, J.P., Jeffrey, R.C.: *Computability and Logic*, forth edition, Cambridge: Cambridge University Press.

[Boker & Dershowitz (2008)] Boker, U. and Dershowitz, N.: 'The Church-Turing Thesis over Arbitrary Domains', in: Avron, Dershowitz, Rabinovich (eds.), *Pillars of Computer Science*, Springer, Berlin, 199-229.

[Brun(2003)] Brun, G. (2003). *Die richtige Formel. Philosophische Probleme der logischen Formalisierung.* Frankfurt: Ontos.

[Dreben (1979)] Dreben, B., Goldfarb, W.D.: *The Decision Problem. Solvable Classes of Quantificational Formulas*, Addison-Wesley, London.

[Floyd (2005)] Floyd, J.: 'Wittgenstein's Philosophy of Logic and Mathematics', in: Shapiro, Stewart (ed.), *The Oxford Handbook of Philosophy of Mathematics and Logic*, Oxford: Oxford University Press, 75-128.

[Frascolla (1994)] Frascolla, P.: *Wittgenstein's Philosophy of Mathematics*, Routledge, London.

[Gödel (1986)] Gödel, K. 1986: 'On formally undecidable propositions of *Principia Mathematica* and related systems I', *Collected Works Volume I Publications 1929-1936*, Oxford UP, Oxford, 144-195.

[Griss (1946)] Griss, G.F.C.: 'Negationless intuitionistic mathematics I', *Koninklijke Nederlandse Akademie van Wetenschappen, Proceedings of the section of sciences 49*, 1127-1133.

[Gumański (1988)] Gumański, L.: 'Remarks on Cantor's Diagonal Method and Some Related Topics', in B. Dyankov (ed.): *Types of Logical Systems and The Problem of Truth (Logic and its Application)*, 45-56.

[Gumański (2000)] Gumański, L.: 'The Decidability of the First-Order Functional Calculus', *Ruch Filosoficzny* 57(3/4), 411-438.

[Gumański (2008)] Gumański, L.: 'A New Proof of Decidability of First-Order Functional Calculus', *Ruch Filosoficzny* 65(3), 419-437.

[Hilbert & Bernays (1970)] Hilbert, D., Bernays, P.: *Grundlagen der Mathematik*, Band 1, Springer, Berlin.

[Jacquette (2004)] Jaquette, D.: 'Diagonalization in Logic and Mathematics', in Gabbay & Guenthner (eds): *Handbook of Philosophical Logic 11*, 2nd edition, Springer, Dordrecth, 55-148.

[Kreisel (1967)] Kreisel, G.: 'Informal rigour and completeness proofs', in Lakatos (ed.): *Problems in the Philosophy of Mathematics*, Amsterdam, North-Holland, 138-171.

[Kvasz (2008)] Kvazs, L. 2008: *Patterns of Change. Linguistic Innovations in the Development of Classical Mathematics*, Birkhäuser, Basel.

[Lampert (2017a)] Lampert, T. 2017: 'Minimizing disjunctive normal forms of pure first-order logic', *The logic Journal of the IGPL*, 25 (3), 325-347.

[Lampert (2017b)] Lampert, T. 2017: 'Wittgenstein's ab-notation: An Iconic Proof Procedure', *Journal of the History and Philosophy of Logic*, 38.3, 239-262.

[Lampert (2018a)] Lampert, T. 2018: 'Wittgenstein and Gödel: An Attempt to make "Wittgenstein's Objection" reasonable', *Philosophia Mathematica 25.3*, 324-345.
[Lampert (2018b)] Lampert, T.: 'Iconic Logic and Ideal Diagrams: The Wittgensteinian Approach', in Chapman, P., Stapleton, G., Moktefi, A., Perez-Kriz, S. and Bellucci, F. (eds), *Diagrammatic Representation and Inference*, Cham, Springer, 2018, 624-639.
[Lampert (2019a)] Lampert, T.: 'Wittgenstein's Conjecture', Mras, Ritter, Weingartner (eds.) *Proceedings of the 41st International Wittgenstein Symposium 2018, Philosophy of Logic and Mathematics*, DeGruyter, 443-462.
[Lampert (2019b)] Lampert, T. 2019: 'A Decision Procedure for Herbrand Formelae without Skolemization', https://arxiv.org/abs/1709.00191, 1-30.
[Landini (2007)] Landini, G.: *Wittgenstein's Apprenticeship with Russell*, Cambridge: Cambridge University Press.
[Marion (1998)] Marion, M.: *Wittgenstein, Finitism and the Foundation of Mathematics*, Clarendon, Oxford.
[Matzer (2018)] Matzer, M.: *Die Unentscheidbarkeit der Quantorenlogik. Widerlegung des Vorschlags für ein Entscheidungsverfahren von Leon Gumański*, unpublished dissertation.
[Meschowski (1967)] Meschowski, H.: *Probleme des Unendlichen. Werk und Leben Georg Cantors*, Springer, Wiesbaden.
[Mycka & Rosa (2018)] Mycka, J, Rosa, W.: 'Związki problemu nierozstrzygalności logiki predykatów pierwszego rzędu z zagadnieniem złożoności' in Myrowski, R., Woleński, J. (eds): *Problemy filozofii matematyki i informatyki*, Wydawnictwo Naukowe UAM, 191-204.
[Nakano (2019)] Nakano, A.: 'Anti-realism and anti-revisionism in Wittgenstein's philosophy of mathematics: a contribution to an exegetical issue', unpublished.
[Peregrin and Svoboda(2017)] Peregrin, J. and V. Svoboda (2017). *Reflective Equilibrium and the Principles of Logical Analysis. Understanding the Laws of Logic*. New York: Routledge.
[Potter (2009)] Potter, M.: *Wittgenstein's Notes on Logic*, Oxford: Oxford University Press.
[Rescorla (2007)] Rescorla, M.: 'Church's thesis and the conceptual analysis of computability', *Notre Dame Journal of Formal Logic* 48(2), 253-280.
[Rodych (1999)] Rodych, V.: 'Wittgenstein's Inversion of Gödel's Theorem', *Erkenntnis* 51, 173-206.
[Shapiro (1982)] Shapiro, S.: 'Acceptable notation', *Notre Dame Journal of Formal Logic*, 23(1), 14-20.
[Simmons (1993)] Simmons, K.: *Universality and the Liar. An Essay on Truth and the Diagonal Argument*, Cambridge University Press, Cambridge.
[Smith (2007)] Smith, P.: *An Introduction to Gödel's Theorems*, first edition, Cambridge University Press, Cambridge.
[Smith (2013)] Smith, P.: *An Introduction to Gödel's Theorems*, second edition, Cambridge University Press, Cambridge.
[Turing (1936)] Turing, A.: 'On Computable Numbers, with an Application to the Entscheidungsproblem', in *Proceedings of the London Mathematical Society* 2 (42), pp. 230-265.
[Quine (1982)] Quine, W.V.O.: *Methods of Logic*, Fourth Edition, Harvard University Press, Harvard, MA.
[Weihrauch (2000)] Weihrauch, K.: 'Computable Analysis', Springer, Berlin, Heidelberg.
[Wittgenstein (1967)] Wittgenstein, L.: *Remarks on the Foundations of Mathematics* (RFM), Massachusetts: M.I.T. Press.
[Wittgenstein (1971)] Wittgenstein, L.: *Prototractatus* (PT), London: Routledge.
[Wittgenstein (1974)] Wittgenstein, L.: *Philosophical Grammar*, Oxford: Blackwell.
[Wittgenstein (1975)] Wittgenstein, L.: *Philosophical Remarks* (PR), Chicago: Chicago Press.
[Wittgenstein (1994)] Wittgenstein, L.: *Tractatus Logico-philosophicus* (TLP), London: Routledge.
[Wittgenstein (1997)] Wittgenstein, L.: *Cambridge Letters* (CL), Oxford: Blackwell.