Article

Explaining the Undecidability of First-Order Logic

Timm Lampert [1,†] and Anderson Nakano [2,†] *

Abstract: Turing proved the unsolvability of the decision problem for first-order logic (*Entscheidungsproblem*) in his famous paper *On Computable Numbers, with an Application to the Entscheidungsproblem*. Similar to decision problems for Turing machines, e.g., the halting problem, and as has been common for undecidability proofs in metamathematics since Gödel, his proof is based on the diagonal method and on hypothetical reasoning. Such proofs, although sound, do not explain to the software engineer why an attempt to specify a solution for the *Entscheidungsproblem* through pattern detection in automated theorem proving (ATP) fails. We provide such an explanation by demonstrating that a finite proof sequence for a provable formula can share all of its inference steps except the last nth step with the first $n-1$ steps of an infinite sequence of inference steps for an unprovable formula in ATP, where both sequences are governed by the same pattern. This pattern, however, only repeats endlessly in the second case. We illustrate this by an example that we generate by mimicking computable sequences for a certain kind of universal Turing machine, namely, splitting Turing machines (STMs), via sequences of inference steps in ATP. This allows us to transfer the straightforward insight that the halting problem cannot be solved through pattern detection to the case of the *Entscheidungsproblem*.

Keywords: Automated Theorem Proving; Entscheidungsproblem; Pattern Detection; Method of Saturation; Halting Problem

1. Introduction

Since [16], textbooks in computer science have repeated Turing's strategy for proving the undecidability of first-order logic (FOL). This proof consists of expressing a Turing machine as an FOL formula and demonstrating that the decidability of FOL implies the decidability of some problem that is unsolvable for Turing machines. This latter problem, in turn, is usually proven to be unsolvable by means of the diagonal method and hypothetical reasoning. Such a proof is independent of any concrete decision method. It proves that there cannot exist any algorithm that solves, e.g., the halting problem by referring to the very special case of self-application. This strategy proceeds as follows. Suppose that there exists a machine H that solves the halting problem. Furthermore, consider a machine M that applies H and, after having done so, reverses the answer of H. If M is run with its own description number as input, this results in a contradiction, which proves the falsity of the initial supposition.

The power of this strategy, combined with its simplicity and its use of idiosyncratic cases of self-application, has the flavor of a "magic trick" ([15], p. 81), and its relation to paradoxes has already provoked qualms on the part of its inventors (see, e.g., [4], p. 151, fn. 5, or the following quote from [16], p. 246). Turing mentions reservations against a proof based on diagonalizing Turing machines to prove a contradiction:

> This proof, although perfectly sound, has the disadvantage that it may leave the reader with a feeling that "there must be something wrong".

To evade these concerns, Turing offers a different proof that Floyd has dubbed the "'Do-What-You-Do Machine' argument" (see [6], p. 130, pp. 129-133, and [5] for a reconstruction of the argument). This argument circumvents the need for a machine that reverses the decision of a machine contained within it. However, it is still based on the diagonal method and on hypothetical reasoning, this time resulting not in a contradiction but in a senseless, tautologous instruction.

check for updates

Citation: Lampert T.; Nakano, A. Explaining Undecidability. *Preprints* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

We do not wish to enter a discussion of a proof strategy based on diagonalization, hypothetical reasoning and the expression of Turing machines by means of FOL formulas, nor do we wish to speculate regarding the extent to which purported doubts concerning its correctness can be overcome (see [8] for a recent discussion on this topic). Instead, we are concerned with the lack of explanatory power of proofs based on this strategy in relation to concrete questions about decision procedure design. In our opinion, part of the dissatisfaction with proofs based on this strategy arises from the unsatisfied desire to come to understand more than the mere fact that certain decision problems cannot be solved since this would imply contradictory or tautologous instructions in the diagonal case. In contrast, we intend to explain why a certain *prima facie* promising method to solve the decision problem for FOL based on pattern detection in automated theorem proving (ATP) fails. In our view, explaining undecidability by invalidating *specific algorithms* that allegedly solve decidability is, in fact, what is needed in software engineering. To our knowledge, such explanations are rarely, if ever, given. We wish to change this situation and thus contribute to a better understanding of the limits of ATP in particular and of decidability in general.

We first expound our question in section 2 before we then present our main argument in section 3. The details of this argument are spelled out in a computer program introduced in section 4. Finally, we conclude with a discussion of our results and an outline for further research in section 5.

2. Expounding the question

When designing algorithms for automated reasoning, one of the goals of a software engineer is to work out specific decision procedures that are as powerful as possible. In this context, a logic programmer is not concerned with the diagonalization of a hypothetical, unreal decision procedure. Instead, she is interested in coming to understand the reasons for limits of actual attempts to spell out concrete decision procedures for FOL formulas independent of hypothetical diagonal cases. The metalogical literature, however, has remained basically concerned with distinguishing decidable from undecidable fragments of FOL independently of concrete proof search methods. In doing so, undecidable fragments are reduced to expressing problems within FOL that are undecidable due to hypothetical reasoning and diagonalization (cf. [1]). For such undecidable fragments, this does not provide much insight regarding the possibilities and limitations of specific methods for making progress in deciding formulas of this class. For decidable fragments, their decidability need not even be proven by some reasonable decision procedure based on decision criteria (see the following paragraph).

Consider, e.g., finite sets of first-order formulas. From a metalogical and classical point of view, we are told that any finite set of formulas is decidable (cf. [3], p. 1). This makes evident the classical, extensional point of view in metalogic. From this point of view, what is asked is whether some decision procedure *exists* rather than *how to specify* a reasonable procedure. Since for any finite set of formulas, there exists a table with the correct entries 1 and 0 for "provable" and "unprovable" formulas, respectively, there exists a computable function that assigns 1 and 0 to each formula. In the case of finite sets of formulas computation is, thus, equivalent to "looking up the answer in a table" ([1], p. 239). However, the logic programmer is concerned with developing a program to generate such a table by applying decision criteria. For her, the mere demonstrable existence of such a table, without the means to construct it by applying a decision criterion, is irrelevant.

Therefore, from the perspective of the logic programmer, the relevant question is the extent to which specifying decision criteria is possible for arbitrary, finite or infinite, sets of formulas. From a classical point of view, one may be content with undecidability proofs proving nothing but the absurdity of supposing the existence of a general decision algorithm, independent of considering any decision criteria. However, from the point of view of a programmer, the more significant question concerns the possibility and limits of specific decision criteria. Roughly speaking, the question of undecidability is not an extensional one but rather an intensional one when considered from this perspective.

Since every provable first-order formula can be decided as provable (a property known as the semidecidability of FOL) and since every first-order formula with finite models can be decided as satisfiable (and, thus, not refutable), the challenging, and still countably infinite, set of formulas is the set of formulas with only infinite models. Formula (1) is a simple example of such a formula. (2) is its clause form with the literals numbered. Interpreting Pxy by $x < y$ in the natural numbers provides an infinite model of (1) and (2); see [1], p. 33, for the proof that (1) has only infinite models.[1]

$$\text{FOL formula:} \quad \forall x_1 \exists y_1 \neg P x_1 y_1 \wedge \forall x_2 (P x_2 y_1 \vee \neg P x_2 x_1) \wedge \forall x_3 P x_3 x_3 \tag{1}$$

$$\text{clause form:} \quad \begin{gathered} \{\{\neg P[x_1, sk_1(x_1), 1]\}, \{P[x_3, sk_1(x_1), 2], \\ \neg P[x_3, x_2, 3]\}\}, \{P[x_4, x_4, 4]\}\} \end{gathered} \tag{2}$$

Since no finite models are available, model finders provide no assistance for an algorithmic account of formulas with only infinite models. For some fragments of FOL, mere inspection of normal forms is sufficient to decide the refutability of an initial formula. For example, monadic FOL, which includes propositional logic, or so-called Herbrand formulas, which lack disjunction in negated normal form, can be decided without employing an exhaustive proof search within a complete calculus. For a simplest example, consider a disjunctive normal form (DNF) of a propositional formula ϕ: ϕ is refutable if and only if each disjunct of its DNF contains a literal A and $\neg A$.[2]

However, pure inspection of normal forms or, more generally, of FOL-formulas or clauses, is of little help with regard to formulas that lack the finite model property. While finite models may be read off from properties of a finite proof search, infinite models may only correspond to an infinite proof search. Refutable formulas, which have neither finite nor infinite models, exist that are only refutable if some rule is applied that iteratively increases complexity. Examples of such a rule are employing clauses repeatedly in resolution or tableaux (cf. the proof search corresponding to Figure 5 below) or applying $A \vdash A \wedge A$ or $A \vdash A \vee A$ in a complete calculus of pure FOL (cf. the proof search corresponding to Figure 6 below). In this case, one cannot read off the logical property in question, such as refutability, from some normal form expression. Instead, one may be obliged to iteratively increase the complexity of the formula to a certain level to find a proof of refutability. This leads to the problem of how to specify the extent of the complexity increase and, thus, the extent of the need to apply rules such as $A \vdash A \wedge A$ before one can decide provability of refutation. Iterative application of such a rule may lead to a proof of a refutable formula or may indicate that no finite model can be inferred from the proof search of a formula that has only infinite models.

The most direct and promising method of deciding at least some formulas with infinite models is the so-called "method of saturation". This method consists of a systematic proof search within a complete calculus that finds a proof in the case of provability, whereas it may terminate in the case of unprovability due to exhaustive application of the rules of the calculus. The problem for the logic programmer is to define criteria that specify *exhaustive rule applications* such that one can conclude that no proof will be found through any further application. The most direct criterion for this purpose is the so-called *criterion of regularity*. If a sequence of inference steps derives the same formula *twice* on a proof search path, then there is no need to continue to search for a proof on this path within an exhaustive search for proofs of minimal length. By this criterion, for example, the set of formulas that can be converted into prenex normal forms with no existential quantifier in

[1] Cf. [12] for a procedure to generate formulas with infinite models only.

[2] [9] demonstrates how Herbrand formulas can be decided without employing a rule that may increase complexity to an arbitrary level in a complete proof search.

the scope of a universal quantifier can already be decided in tableau or resolution calculi. This set of formulas, however, does not contain formulas with only infinite models. As soon as existential quantifiers occur in the scope of universal quantifiers – as it is the case in formulas with infinite models only – new variables occur in the iterative application of an inference rule in ATP and, thus, regularity does not suffice to terminate endless iterations. To do so, some generalization of this criterion is needed.

We distinguish two senses of *regularity* and, thus, *regular sequences*: narrow and broad. The former implies a repetition of members in a computable sequence in the strict sense of repeating exactly the same expression (as is the case according to the standard regularity criterion), whereas the latter implies the repetition of a certain *pattern* in a computable sequence that can be identified by a *law*.

By a *law*, we mean a rule that generates, without further computation,[3] potentially infinitely many members of a sequence by generating the nth member either directly from previous members (inductive definition) or directly from n (explicit definition). Examples of sequences that can be generated by a law are $b, ab, aab, aaab, aaaab, ...$, generated by the regular expression "$a * b$"; the sequence of Fibonacci numbers, $0, 1, 0+1, 1+(0+1), (0+1)+(1+(0+1)), (1+(0+1))+((0+1)+(1+(0+1))), \ldots$, generated by $a_n = a_{n-1} + a_{n-2}$; and the sequence of squares, $1 \cdot 1, (1+1) \cdot (1+1), (1+1+1) \cdot (1+1+1), \ldots$, generated by $a_n = n \cdot n$. An example of a sequence that could not hitherto be generated by a law is the computable sequence of prime numbers. According to our understanding, this sequence, although computable, seems not to be governed by a pattern that could be identified by a law. Instead of *constructing* the next prime number, we must *search for* it in a finite interval. Since this interval is finite, we can compute the next prime number; however, the results of these searches are not governed by a law. As a result, the sequence of prime numbers does not constitute a regular sequence governed by a pattern in our sense.

Table 1 presents examples in number theory comparing the irregular decimal expansions of $\sqrt{2}$ and $\frac{\pi}{4}$ to their so-called regular ($\sqrt{2}$) and irregular ($\frac{\pi}{4}$) continued fractions, which can be characterized as regular sequences in the narrow and broad senses, respectively. Note that the usual distinction between regular and irregular *continued fractions* does not correspond to our distinction between regular and irregular *sequences*. Instead, it refers to the partial numerators, which are always 1 in the case of regular continued fractions, while they vary in the case of irregular continued fractions. Since the partial numerators are always 1 in regular continued fractions, they are ignored in shorthand notation. Therefore, the shorthand notation for the regular continued fraction for $\sqrt{2}$ is $[1; 2, 2, 2, \ldots]$, which makes evident its regularity in the narrow sense. The irregular continued fraction for $\frac{\pi}{4}$, however, is a regular expansion in the broad sense since the numerators as well as the denominators are not strictly identical but develop by a law.

Henceforth, we will use the unqualified phrase "regular sequence" to refer to a "regular sequence in the broad sense", which is a "sequence generated by a law". Furthermore, whenever we speak of a "pattern of a sequence", we presume that this pattern can be specified by a law. However, this does not imply that this pattern is, in fact, endlessly repeated in a computable sequence; as we will see, this may or may not be the case. That is, we also allow only a part of a finite or infinite sequence to be defined by a law: in this case, n in the inductive or explicit definition is, in fact, restricted to a finite number. The pattern itself is always finite but can be repeated either a finite number of times or indefinitely. Finally, when we speak generally of "rules" or "instructions of Turing machines", "Turing machines" or "computable sequences", we do not presume that they are or can be specified by laws. Computable sequences may involve lawless parts or they may involve law-governed finite sequences that may be both (i) proper parts of a finite computable sequence or (ii) endlessly repeating parts of an infinite computable sequence. We will argue

[3] Note that translations into other notations include further computation. This is why the following sentence in the main text does not present the sequence of Fibonacci numbers or the sequence of squares in the decimal notation, which would translate regular sequences into irregular sequences.

Number	Narrow Regularity	Broad Regularity	Irregularity
$\sqrt{2}$	$1+\frac{1}{2+\frac{1}{2+\frac{1}{\ddots}}}$,	–	$1.4142135\ldots$
$\frac{\pi}{4}$	–	$1+\frac{1^2}{(1+(1+1))+\frac{(1+1)^2}{((1+(1+1)+(1+1))+\frac{(1+1+1)^2}{\ddots}}}$	$0.7853981\ldots$

Table 1: Irregular and regular number sequences

that this is relevant to question of the possibility to decide provability based on patterns in ATP.

Regularity in the narrow sense is an example of a simplest repeating pattern that, in fact, allows one to terminate a sequence of inference steps in the case of unprovability. If regularity applies to all proof paths, then the unprovability of the initial input formula can be decided according to the saturation method within an exhaustive proof search. An example of a simple formula that can be decided based on regularity is (3), with its clause form given in (4).

$$\text{FOL formula:} \quad \exists y_1 \neg P y_1 y_1 \wedge \exists y_2 \forall x_1 \forall x_2 ((P x_1 x_2 \vee \neg P y_2 x_2) \wedge P y_2 y_2) \tag{3}$$

$$\text{clause form:} \quad \begin{array}{c} \{\{\neg P[sk_1, sk_1, 1]\}, \{P[x_2, x_1, 2], \\ \neg P[sk_2, x_1, 3]\}, \{P[sk_2, sk_2, 4]\}\} \end{array} \tag{4}$$

Figure 1 shows the application of the regularity criterion in the tight connection tableau calculus initialized by clauses with only negative literals. This calculus is known to be complete (cf. [13]). We presume this calculus in this paper for ATP for clause forms.[4] Moreover, we restrict ourselves to cases in which the proof tree has only one node (i.e., one tableau), that is, to cases in which a *deterministic* proof search is known to be complete. Such a proof search implies that all proofs considered are of minimal length. This is a significant simplification for our reasoning because it allows us to abstain from considering the relevance of our criteria for a mere elimination of nonminimal proofs and to focus instead on the validity of the purported decision criteria.

To focus on questions of pattern detection, we visualize the steps of rule application by means of color diagrams, similar to what is done in the case of, e.g., cellular automata (cf. [17]). Different states of Turing machines or predicates (relation signs) of logical formulas are represented by different colors in the first position on each line. Different symbols at positions on the tapes of Turing machines or at positions of predicates are represented by different colors in the corresponding positions (after the first) on each line. The same is true for the positions of Skolem functions. Different colors depict different Skolem functions occurring in nested Skolem functions which occur at a position of a predicate. For simplicity, we represent only the relevant main branch of maximal length in the color diagrams of our deterministic tableau proofs.

Examples indicate that regularity in the narrow sense can be generalized to cases in which the proof search runs in endless loops but without repeating expressions in the narrow sense. As Figure 2 illustrates for the simple clause (2) with infinite models only, the proof search may fall into an obvious endless nesting scenario. We can identify a law that tells us how the sequence of formulas develops from the sequence alone, without considering the clauses or instructions of the proof search algorithm. In this case, the sequence of inference steps is, in fact, governed by a repeating pattern, although without

[4] Since the focus of our paper is the principal limitation of pattern detection in ATP, we abstain from specifying the technical details of ATP. The interested reader may consult the pertinent paper [13] and our implementation for details (cf. section 4).

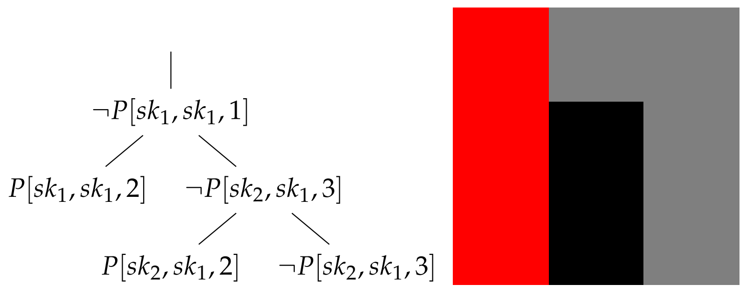

Figure 1. Deciding unprovability by means of the regularity criterion in ATP for (4)

strict repetition of the same expression. The question is the extent to which we can, in fact, infer from the finite repetition of a pattern that it will repeat endlessly.

There are, of course, more complicated cases of endless looping than the one shown in Figure 2, and the proof search may become too messy for a human to identify any pattern at all. However, this may be a problem that one may hope to overcome by means of translation into other, more perspicuous notations, or it may be a problem of human pattern identification capabilities that one may hope to overcome by means of machine learning. Thus, one may strive for intricate pattern detection for endless loops in ATP. Our question is how to explain to the logic programmer why this endeavor is hopeless. Our philosophical motivation for this question is to explain why a conception of logic that claims that logical properties, such as provability, are reducible to pattern detection in a suitable notation fails when applied to the whole realm of FOL (cf. [10] for details about this conception of logic, which is based on insights from Wittgenstein's early work). We maintain that this conception works only for normal form transformations of *fragments* of FOL (see p. 3 above); we intend to show that the reducibility claim is incorrect when applied to a general proof search in FOL.

We call a purported general decision criterion referring to a *finite part* of a computable sequence, where this finite part repeats a pattern, a "loop criterion". This criterion is used to support an inference from finite repetitions to endless repetitions and, thus, to determine unprovability. It generalizes the regularity criterion. We intend to answer the following question by providing concrete examples: Why is it impossible to generalize the regularity criterion in order to correctly determine unprovability by means of a general loop criterion in a complete and automated proof search?

To our knowledge, this question has not been raised in the literature to date. We will address this question by considering a simplest undecidable class of FOL formulas, namely, so-called Krom–Horn clauses, and their translation into pure FOL formulas. Krom–Horn clauses are clauses with at most two members, at most one of which is negated. We will consider only sets of Krom–Horn clauses that result from translations of so-called splitting Turing machines (STMs). We specify STMs for the purpose of making evident the limitations of a decision criterion specified in terms of a loop criterion. Translating STMs into a special sort of Krom–Horn clauses is sufficient to illustrate how the halting problem for STMs transforms into the decision problem for the corresponding Krom–Horn clauses. The deterministic search for proofs of Krom–Horn clauses and of their corresponding pure FOL translations will mimic the behavior of STMs. However, we do not relate the proof search in FOL to STMs in order to prove undecidability by *expressing* STMs in FOL (as is usually done in undecidability proofs), nor do we make use of the diagonal method. Instead, we mimic the execution of deterministic STMs via a deterministic proof search to come to understand the impossibility of specifying a general criterion for detecting endless loops within an exhaustive proof search in FOL. Our discussion will not only dispense with the diagonal method in combination with hypothetical reasoning but also abstain from *expressing* Turing machines by means of FOL formulas. It will be purely syntactic and, thus, demonstrable by our implementation of the translation and proof search procedures.

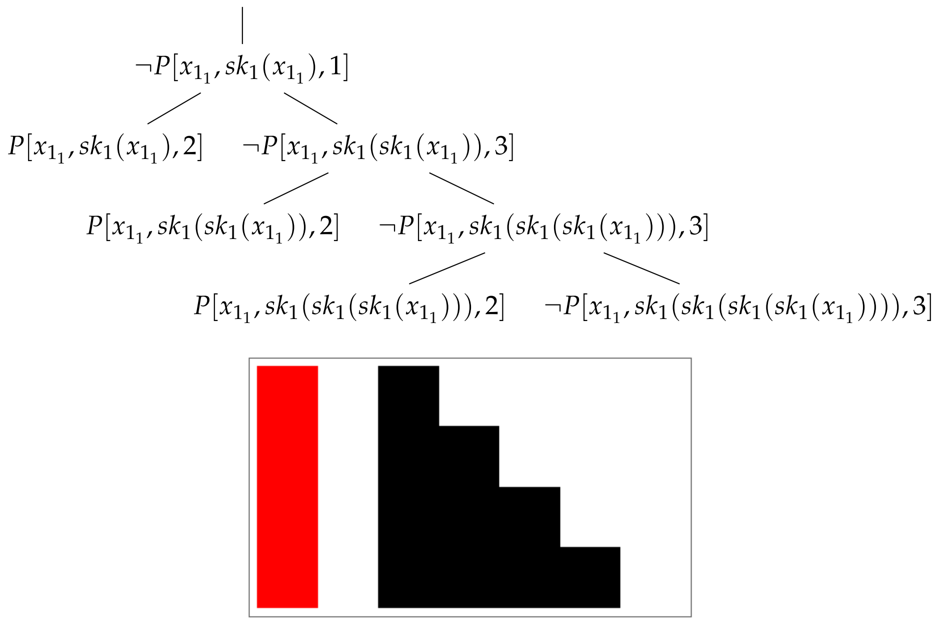
$\neg P[x_{1_1}, sk_1(x_{1_1}), 1]$
$P[x_{1_1}, sk_1(x_{1_1}), 2]$
$\neg P[x_{1_1}, sk_1(sk_1(x_{1_1})), 3]$
$P[x_{1_1}, sk_1(sk_1(x_{1_1})), 2]$
$\neg P[x_{1_1}, sk_1(sk_1(sk_1(x_{1_1}))), 3]$
$P[x_{1_1}, sk_1(sk_1(sk_1(x_{1_1}))), 2]$
$\neg P[x_{1_1}, sk_1(sk_1(sk_1(sk_1(x_{1_1})))), 3]$

Figure 2. Endless nesting in ATP for (2)

3. Results: From the halting problem to the *Entscheidungsproblem*

We first introduce STMs and explain why it is hopeless to attempt to solve the halting problem for STMs based on patterns in their evolution (section 3.1). As the main content of our paper, we then explicate how this explanation transfers to the *Entscheidungsproblem* by considering a proof search for clauses with Skolemization (section 3.2) and for pure FOL formulas without Skolemization (section 3.3).

3.1. Splitting Turing machines

An STM is an automaton that has a circular tape with cells that can be split. The specifications and behavior of STMs are defined as follows.

Def. (Splitting Turing machine, STM): An STM is described by a tuple $S = (Q, \Sigma, f, q_1, q_f, c_n)$, where Q and Σ are the finite sets of states and tape symbols, respectively; $q_1 \in Q$ is the initial state; $q_f \in Q$ is the halting state; c_n is the initial tape of size n that defines the symbols for all n initial positions of the machine; and the transition function f is undefined for the state q_f but is defined for all $q \in Q, q \neq q_f$.

We write f as a list of transition rules. Each transition rule is expressed as a quadruple $t = (q_x, \sigma, v, q_y)$, with initial state q_x, read symbol σ, instruction v, and next state q_y. The possible instructions are as follows:

W*s***:** write the symbol s, $s \in \Sigma$;

S: split the current cell, duplicating its content;

L: move the scanner counterclockwise;

R: move the scanner clockwise.

STMs are universal machines because they can simulate clockwise Turing machines (CTMs), which are universal; cf. [14], pp. 107-109.[5] Therefore, if the halting problem is solvable for STMs, then it is solvable for all Turing machines.

[5] It is trivial to show that CTMs can be simulated by STMs. We need only consider that i) CTMs are automata that move to the right after every instruction and either write one symbol on the tape or split one cell of the tape and write two symbols in the split cells and ii) all of these operations are available in STMs.

It is a well-known feature that even very simple Turing machines may induce rather irregular sequences in their evolution; cf., e.g., Figure 3 for the 5-state, 3-symbol STM described by (5) (the color diagram evolves from left to right).

$$
\begin{array}{ll}
Q\colon & \{P, Q1, Q2, Q3, H\}, q_1 : P, q_f : H, \\
\Sigma\colon & \{0, 1, 2\}, \\
f\colon & \begin{array}{l}\{\{P, 1, \{L\}, P\}, \{P, 0, \{W, 1\}, Q1\}, \{P, 2, \{R\}, Q3\}, \{Q1, 1, \{R\}, Q2\}, \\ \{Q1, 0, \{R\}, Q2\}, \{Q2, 1, \{W, 0\}, Q1\}, \{Q2, 2, \{L\}, P\}, \{Q3, 1, \{S\}, Q1\}\},\end{array} \\
c_2\colon & \{1, 2\}.
\end{array} \tag{5}
$$

Table 2: A 5-state, 3-symbol STM inducing the irregular color diagram in Figure 3

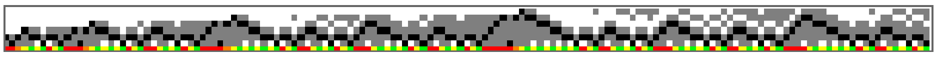
Figure 3. Color diagram of the evolution of the STM described by (5) over 150 steps

One might wonder whether seemingly irregular sequences already indicate that a decision procedure based on pattern detection for endless looping is futile. However, regularity depends on notation. A different notation may well enhance the possibility of solving decision problems[6], e.g., by converting irregular sequences into regular sequences. Approximating a number by means of different sequences in different notations, which can be translated into each other, is one example of this; cf. Table 1. For example, square roots cannot be identified based on any pattern in their representations in decimal notation, but they can be identified based on the *periodicity* of their regular continued fractions. Thus, one may argue that a pattern may indeed be identifiable by translating the irregular sequence generated by an STM into a regular sequence by translating the STM and the sequence it generates. Our strategy of translating STMs into clauses or FOL formulas and generating sequences of inference steps by applying a logical calculus enables the investigation of the relations between sequences in different notations with respect to the resulting patterns. The aforementioned normal form transformations for the purpose of solving decision problems for fragments of FOL illustrate the dependency of the ability to solve decision problems in logic on a notation that may reveal logical properties in the form of common patterns in a suitable notation. From the practical perspective of a software engineer, one might also wonder whether machine learning could achieve better performance than humans and, at least, gain the ability to decide problems to a reliable degree by learning from evolutionary patterns in a training database.

However, one can demonstrate by example that solving the halting problem based on patterns of STM sequences or solving the *Entscheidungsproblem* based on patterns of sequences of inference steps is futile independent of the extent to which irregular sequences can be reduced to regular sequences. We show this by considering *regular* instead of irregular sequences, i.e., by considering cases in which a pattern is *indeed* found. We first do this for sequences computed by STMs and the question of solving the halting problem in order to then ask whether the same approach can be extended to sequences in ATP and the *Entscheidungsproblem*.

Our example, in which STM1 and STM2 are specified as shown in Table 3 and their color evolution diagrams are presented in Figure 4, demonstrates that some repetition of a regular pattern does not allow one to decide that a Turing machine is nonhalting. STM2 differs from STM1 only by the replacement of one of the halting instructions with a nonhalting instruction. The evolution diagrams do not differ through the first 78 steps, and these steps already contain one complete repetition of the regular pattern, which

[6] Cf. [11] for a general discussion of this claim.

Q:	$\{P, P1, P2, P3, P4, H\}, q_i : P, q_f : H,$
Σ:	$\{1, 2, 3, 4\},$
f:	$\{\{P, 1, \{W, 2\}, P1\}, \{P, 2, \{R\}, P\}, \{P, 3, \{R\}, P4\},$
	STM1:$\{\mathbf{P}, \mathbf{4}, \{\mathbf{W}, \mathbf{4}\}, \mathbf{H}\}$, **STM2:**$\{\mathbf{P}, \mathbf{4}, \{\mathbf{W}, \mathbf{4}\}, \mathbf{P1}\}$,
	$\{P1, 1, \{R\}, P1\}, \{P1, 2, \{R\}, P1\}, \{P2, 3, \{R\}, P\}, \{P1, 4, \{R\}, P\},$
	$\{P2, 1, \{R\}, P2\}, \{P2, 2, \{W, 1\}, H\}, \{P2, 3, \{R\}, H\}, \{P2, 4, \{R\}, P3\},$
	$\{P3, 1, \{S\}, P\}, \{P3, 2, \{S\}, H\}, \{P3, 3, \{R\}, H\}, \{P3, 4, \{W, 1\}, H\},$
	$\{P4, 1, \{R\}, P4\}, \{P4, 2, \{W, 1\}, P4\}, \{P4, 3\{R\}, P2\}, \{P4, 4, \{R\}, P4\}\},$
c_5:	$\{1, 3, 1, 1, 4\}.$

Table 3: STM1 (halting) and STM2 (not halting), differing by one instruction

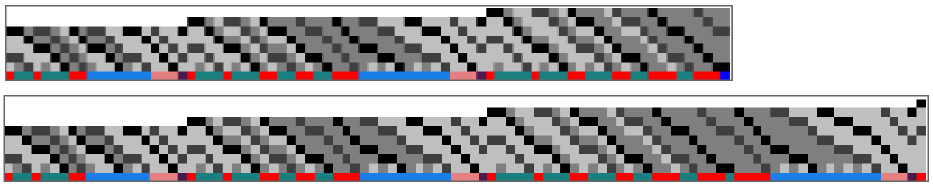

Figure 4. Color diagrams for STM1 and STM2 up to 100 steps

merely increases in its level of nesting. In the next step, however, STM1 halts, while STM2 continues forever in a regular manner.

It is easy to describe the difference of the functions that the instructions of STM1 and STM2 implement. Both take as input two sequences of 1s, a and b, separated by the symbols 3 and 4. While a is increased by 1 in each loop, b remains constant. STM1 halts as soon as $a > b$, while STM2 goes on with increasing a forever. Yet, describing the implemented functions does not amount to provide a general decision criterion for distinguishing finite from infinite loop processes. Recall that, from the point of view of metalogic, it is possible to mechanically decide for the members of a *finite* set of STMs, including STM1 and STM2, whether they halt or do not halt. One may even devise an algorithm that decides, for an infinite set of STMs that includes STM1 and STM2, whether its members halt or do not halt. This can be done by considering special forms of machines, e.g., machines that implement Do-until loops that do not modify the loop counter during the loop, but only increment or decrement the counter after each loop until a certain value is reached and, additionally, each single loop is known to end after a finite number of steps. Yet, what is in question is a decision criterion that applies to *all* STMs and, thus, distinguishes finite from infinite loop processes *in general*. Our pair of machines, STM1 and STM2, is an arbitrary pair of STMs that illustrates the principal problem of defining a *general* criterion for escaping a loop. In contrast to the hypothetical diagonal case, such a pair consists of concrete STMs. This makes it possible to translate the problem to a factual pair of expressions within FOL and to illustrate it by real proof search.

For our argument it is important to understand why we investigate a loop criterion that refers to nothing but the evolution of the tape of the machine without additionally taking the instructions into account. To this end, we shall discuss why the information provided by the instructions is not helpful to our problem. We will first discuss the case where the instructions are taken into account independently of their application, i.e., independently of the evolution of the content of the machine's tape (what we call in brief the *extension* of the instructions), then we will discuss the case where the instructions are taken into account in the course of the development of their extension.

In the first case, we note that the halting problem cannot be solved by some property of the *bare* instructions. By such a property we mean a computable property of the instructions that does not make use of information extracted from their extensions. There are indeed properties in this sense that distinguish *some* nonhalting STMs from halting STMs, e.g. the property that no halting instructions are part of the definition of an STM at all or the property that the states or the symbols contained in the halting instructions never occur in the other instructions. Yet, these properties are not common to *all* nonhalting STMs. The problem to distinguish halting from nonhalting STMs is the same as the problem to distinguish provable from unprovable formulas by properties of the formulas themselves or of some normal form. Such properties provide sufficient but no necessary criteria, because there are cases where the iterated application of rules is needed (i.e., there are cases where the development of the extension of the instructions is necessary). STMs may involve splitting of cells within loops that iterate the application of certain rules. A solution of the halting problem has to distinguish STMs like STM1 that escape a loop with splittings from STMs like STM2 that never escape a loop. However, the properties of the bare instructions do not refer to the number of applying rules iteratively and, thus, do not contain a general criterion for this distinction. If at all, a criterion must apply to the extension resulting from *applying* rules.

In the second case, we first note that a distinction between finite and infinite loops cannot be inferred from "new" instructions that are not applied within the looping process plus some sort of *backward evolution*, either. To escape a loop, the condition of a new instruction must be satisfied. In the case of STM1, for example, the looping process is escaped by a configuration with state P and symbol 4 at position 1 on the tape, which results in applying the halting instruction. In the case of STM2, however, the same condition does not escape the loop, because the corresponding instruction neither writes a new symbol on the tape nor reaches a new state that is not part of the loop. These conditions are part of the new instructions; they constitute their first two positions. But this does not suffice to identify previous configurations that must obtain to reach the conditions of the instructions. The reason for this is that the rules R and L do not tell the symbol of the square after the application of the rule. Possible escaping configurations for a given looping process could only be specified without referring to previous configurations by combinatorial means, because the definition of an STM contains no more than an *initial* configuration. The number of possible escaping configurations that one might specify combinatorially, however, is indefinite due to the splitting process. Therefore, it is impossible to compute configurations that will escape the loop backwards from possible new instructions not applied yet in the loop. So, even if one were able to decide for a given configuration whether a looping process will ever reach it, this does not suffice to decide whether the looping process will terminate.

Moreover, the pure definition of an STM plus a finite number of steps of a looping process do not determine a *property* of a configuration that would make it possible to finally reach the condition of a new instruction, either. The definition of an STM only specifies its instructions and its initial configuration and this determines only a stepwise process. The only possibility to infer later configurations from previous ones without going through the whole process step by step consists in identifying a pattern and deducing later steps by assuming that its law determines an endless repetition without any escape.[7] This assumption, however, is what is in question.

That is why we do not consider a purported decision criterion for the halting problem that is based on the pure definitions of Turing machines or on computing escaping configurations from instructions not involved in a loop. Instead, we investigate a loop criterion that refers to nothing but the evolution of applying rules. It is this situation that we want to investigate for ATP regarding the *Entscheidungsproblem*. In doing so, we do assume that it is possible to identify mechanically loops that repeat a certain pattern. In the case of STM1 and STM2, e.g., the looping process starts from configurations that only

[7] This does not mean that we claim to solve the halting problem by considering a complete list of possible attempts to solve it. We merely motivate our focus on the loop criterion by distinguishing it from attempts to solve decision problems without considering looping processes as they evolve. Furthermore, recall that we do not purport to contribute to the discussion of the halting problem. Instead, we intend to contribute to the discussion of the *Entscheidungsproblem* by referring to straightforward insights concerning the halting problem.

differ by the increasing number of 1s in the first sequence a of 1s. Within each loop the same instructions are applied in the same order, only the number of applications of these instructions differ in each repetition due to the increasing nesting. A universal machine can identify the looping process by detecting the successive increase of nesting and the repeating succession of rule applications in consequence of it. Based on these assumptions, the question arises whether it is possible to infer non-halting from the identification of a looping process. We provide a negative answer to this question.

Our argument consists in providing a pair of STMs, STM1 and STM2, that shares the looping process. While, however, STM1 escapes the loop and halts, STM2 repeats the looping process endlessly. All of the halting instructions in STM1 and STM2 except the one missing in STM2 are irrelevant, because their conditions will never be satisfied. STM2 does the same as STM1 with the only difference being that it continues its execution after each comparison of the increasing number a with b. A loop criterion is refuted by the fact that by simply redefining a relevant instruction where a halting condition will be satisfied, one can specify a corresponding machine that will behave likewise, with the only difference being that its execution will continue while the other halts. Thus, examples such as STM1 and STM2 can be generalized to an arbitrary number of pairs of halting and nonhalting machines started with a certain tape configuration.

This allows us to systematically invalidate the following assumption, which is assumed in any specific attempt to formulate a loop criterion based on pure pattern detection:

Assumption 1. *For no computable infinite sequence x that endlessly repeats a pattern ϕ is there a computable finite sequence y that contains a proper finite part z that (i) y shares with x, (ii) is generated by the same rules when computing x and y, and (iii) consists of only a finite number of repetitions of the pattern ϕ in y while this pattern is repeated endlessly in x.*

(ii) is necessary because we are not concerned with the trivial fact that any finite part of a sequence revealing a pattern can be computed by a Turing machine simply by listing the symbols of this finite part. Instead, we are interested in pattern detection within ATP, which is based on a general and complete proof search algorithm that generates sequences of inference steps. Our question is whether supplementing such an algorithm with a loop criterion can make it able to distinguish provable from unprovable formulas. Within the framework of a proof search algorithm, there is no degree of freedom to specify or change rules for generating the sequence of inference steps; all that one can do is terminate the generation of inference steps on the basis of an additional criterion. This is why the relevant assumption refers to the application of the same rules when generating a finite part z that contains a repeating pattern. Note that this does not imply that the *entire* sequences x and y are generated by the same rules, nor that the *entire* sequences x and y are generated by laws. STM1 differs from STM2 by one rule, but this rule is not applied when generating the shared pattern, which alone is identified by a law. This law is able to produce the infinite and regular sequence that STM2 also generates via its rules, but it does not produce the whole finite and *irregular* sequence that STM1 generates via its rules because it does not generate the final step of this latter sequence.

The refutation of the above assumption implies that STM2 can no longer be decided as nonhalting according to a pure loop criterion that is based on nothing but the detection of a pattern within a *finite* part of a computable sequence. Note that, by increasing the second number b against the first number a is compared, we can arbitrarily increase the number of repetitions executed before STM1 comes to a halt. Therefore, a loop criterion cannot be rescued by *simply* increasing the number of repetitions to be considered.

We do not invalidate the assumption by objecting that it may be difficult to identify patterns in a suitable notation but rather by the fact that such a criterion does not identify a sufficient condition for nonhalting in the case that loops can be identified mechanically. In what follows, our intention is to extend this straightforward refutation of the validity of a loop criterion for solving the halting problem to applying such a criterion to solve the *Entscheidungsproblem* within ATP. Here, it is not similarly trivial to see the invalidity of a loop criterion when invoked as a general decision criterion. This is not trivial for the following reasons. (i) The regularity criterion does, in fact, identify a regular sequence in the narrow sense, which indeed allows us to identify satisfiability. Therefore, one can consider the possibility of generalizing this feature to regular sequences in ATP in general. (ii) An endless proof search for certain satisfiable formulas is, in fact, encountered for regular sequences in the broad sense; cf. the clauses in (2) and Figure 2. Therefore, generalizing the regularity criterion to a loop criterion is indeed a promising and intuitive possibility for making progress in deciding FOL formulas. (iii) Since regularity and decidability depend on notation, invalidating a loop criterion for sequences generated by STMs does not *immediately* imply invalidating it for ATP. Our main argument is to demonstrate how the (trivial) impossibility of solving the halting problem based on a loop criterion can, in fact, be transposed to establish the (nontrivial) impossibility of solving the *Entscheidungsproblem* based on a loop criterion.

Furthermore, one typically examines proofs in logic without considering their relation to the execution of Turing machines. One may even have reservations concerning the endeavor to overload FOL formulas with interpretations that go beyond a pure proof-theoretic point of view. Thus, one may be inclined to abstain from *expressing* Turing machines by means of FOL formulas.[8] To circumvent these reservations, we will show in the following that and how the impossibility of specifying a loop criterion for the halting problem can be extended to the impossibility of doing so within ATP without *expressing* STMs within the language of FOL. Instead, we will show how to mimic the behavior of STMs in a deterministic proof search for the rather simple case of Krom–Horn clauses and their translations into pure FOL.

3.2. *Proof search with Skolemization*

We now proceed to show how our explanation of the invalidity of a loop criterion as a decision criterion for the halting problem can be extended to establish the unsolvability of the decision problem for FOL by employing a loop criterion.

We wish to show how to mimic the behavior of STMs in a tight connection tableau calculus. Let us first enumerate the elements we will use for the representation of the machine and the contents of its tape:

1. A Skolem constant sk_G.
2. A Skolem constant sk_s for every $s \in Q$.
3. A unary Skolem function sk_a for every $a \in \Sigma$. We also impose the restriction that, in the clauses, the only allowed argument for these Skolem functions is x_1. This will allow easy translation from clauses to pure FOL.
4. A predicate letter P_s for every $s \in Q$.
5. Some auxiliary predicate letters, to be defined below.

An STM in a certain state and with certain symbols written on its tape will be represented by a literal as follows: the different states of the machine will be represented by different predicate letters (and by different Skolem constants, see 2 above; this redundancy is merely for simplicity); the tape of the machine will be represented by the arguments of the predicates, with the first argument representing the position of the scanner of the machine; the different symbols that may be written on the tape of the machine will be represented by different Skolem functions; and the potentially infinite nature of the tape (due to the splitting operation) will be represented by the potentially infinite nesting of these Skolem functions. The Skolem constant sk_G marks the end (the "ground") of nesting.

From here on, we will use the metavariable α to represent variables that run through the values of Σ and the metavariable β to represent variables that run through the values

[8] Cf. [11], section 5, which criticizes semantic versions of undecidability proofs of FOL. Our purely syntactic version of mimicking the behavior of STMs seems to us a considerable improvement over versions of undecidability proofs that are based on a naive understanding of the soundness of FOL.

of Q. We also stipulate that the first argument of the predicates will always represent only a *single* cell in the machine. The reason for this stipulation is merely practical.

We suppose that the tape is initially filled with $s_1, s_2, \cdots, s_n$, where all $s_i \in \Sigma$. The initial clauses of the FOL formula in clause normal form are as follows:

$$\{\neg F_{aux0}[sk_G, ..., sk_G]\} \tag{6}$$

$$\{F_{aux0}[x_1, \ldots, x_1], \neg P_{q_1}[sk_{s_1}(x_1), sk_{s_2}(x_1), \ldots, sk_{s_n}(x_1), sk_G]\} \tag{7}$$

Note that the tape, which is of size n, is represented by $n+1$ arguments. The $(n+1)$-th argument will always be sk_G, which is useful for the specification of Rule R below.

For the halting state q_f, we add the following unit clause:

$$\{P_{q_f}[x_1, ..., x_n, sk_G]\} \tag{8}$$

Let us now see which clauses are needed for every kind of transition rule. We suppose that the description of each rule is given in the form (q_x, σ_1, v, q_y).

We will skip the translation of Rule L because we do not need it in our examples or for our argument.

3.2.1. Rule W

Suppose that the symbol to be written is σ_2. To represent Rule W, we simply add the following clause:

$$\{P_{q_x}[sk_{\sigma_1}(x_1), x_2, x_3, \ldots, x_{n+1}], \neg P_{q_y}[sk_{\sigma_2}(x_1), x_2, x_3, \ldots, x_{n+1}]\} \tag{9}$$

3.2.2. Rule S

To represent Rule S, we add the following clause:

$$\{P_{q_x}[sk_{\sigma_1}(x_1), x_2, x_3, \ldots, x_{n+1}], \neg F_{auxS_{\sigma_1}}[sk_{\sigma_1}(x_1), x_2, x_3, \ldots, x_{n+1}, sk_{q_y}]\} \tag{10}$$

Moreover, we also add the following clauses to the set of clauses (this set of clauses is added only once, not for every Rule S that determines the behavior of the machine):

$$\{F_{auxS_\alpha}[x_2, x_1, x_3, \ldots, x_{n+1}, sk_\beta], \neg P_\beta[x_2, sk_\alpha(x_1), x_3, \ldots, x_{n+1}]\} \tag{11}$$

3.2.3. Rule R

The specification of Rule R is much more difficult than that of Rules W and S. Let us begin with an example of how mimicking this rule in a tight connection tableau calculus works. Suppose that at a given moment of the machine's execution, its state q_x and the tape $[1, 2, 3, 4, 5, 6]$ are represented by the following literal, which appears on a leaf of a certain open branch of the tableau:

$$P_{q_x}[sk_1(sk_G), sk_2(sk_3(sk_4(sk_G))), sk_5(sk_6(sk_G)), sk_G]$$

The idea is that at the end of the mimicking operations in the tableau, we shall obtain the following literal on the leaf of this branch:

$$P_{q_y}[sk_2(sk_G), sk_3(sk_4(sk_5(sk_6(sk_G)))), sk_1(sk_G), sk_G]$$

representing the tape $[2, 3, 4, 5, 6, 1]$ and the machine in state q_y. This evolution during the application of the rules of the tableau calculus is designed to mimic the fact that the header of the machine moves one position to the right on the tape, changing from the state q_x to the state q_y.

We first must ensure that Rule R will be applied only if the scanned symbol is σ_1. To accomplish this, we add the following clause:

$$\{P_{q_x}[sk_{\sigma_1}(x_1), x_2, x_3, ..., x_{n+1}\}, \neg F_{aux_R}[sk_{\sigma_1}(x_1), x_2, x_3, ..., x_{n+1}, sk_{q_y}]\} \tag{12}$$

Now, we start moving to the right. We first add the following clauses to the set of clauses:

$$\begin{gathered}\{F_{aux_R}[x_2, sk_\alpha(x_1), x_3, \ldots, x_{n+2}],\\ \neg F_{aux1_\alpha}[sk_\alpha(x_1), x_1, sk_G, x_3, \ldots, x_n, x_2, x_{n+1}, x_{n+2}]\}\end{gathered} \tag{13}$$

The rationale for the arguments of F_{aux1_α} will only be made clear after the following operations 1 to 4 are explained. We need to accomplish several things with the arguments of F_{aux1_α} to represent the tape after the execution of the rule:

1. Make the first argument $sk_\alpha(sk_G)$ (for the particular value of α in question).
2. Pop the represented symbols on the tape contained in x_1 (second position of F_{aux1_α}), push them over the third position, and finally remove the second position.
3. Pop the inverted x_1 obtained in operation 2 above and push these symbols over x_3 to put them in the right order, again removing the second position.
4. Use the last argument of F_{aux1_α}, which stores the next state, to finally construct the correct predicate letter.

With respect to clauses (13)–(21), and in contrast to clause (12) above, we note in advance that we do not need to add these clauses for *every* Rule R but instead add them only once.

To accomplish the first task, we add the following clauses:

$$\{F_{aux1_\alpha}[x_2, x_3, \ldots, x_{n+3}, x_1, x_{n+4}], \neg F_{aux2}[sk_\alpha(x_1), x_3, \ldots, x_{n+3}, x_1, x_{n+4}]\} \tag{14}$$

To accomplish the second task, we add the following clauses:

$$\{F_{aux2}[x_2, sk_\alpha(x_1), x_3, \ldots, x_{n+4}], \neg F_{aux3_\alpha}[x_2, x_1, x_3, \ldots, x_{n+4}]\} \tag{15}$$

$$\{F_{aux3_\alpha}[x_2, x_3, x_1, x_4, \ldots, x_{n+4}], \neg F_{aux2}[x_2, x_3, sk_\alpha(x_1), x_4, \ldots, x_{n+4}]\} \tag{16}$$

$$\{F_{aux2}[x_1, sk_G, x_2 \ldots, x_{n+3}], \neg F_{aux4}[x_1, x_2 \ldots, x_{n+3}]]\} \tag{17}$$

The third task is similarly completed by adding the following clauses:

$$\{F_{aux4}[x_2, sk_\alpha(x_1), x_3, \ldots, x_{n+3}], \neg F_{aux5_\alpha}[x_2, x_1, x_3, \ldots, x_{n+3}]\} \tag{18}$$

$$\{F_{aux5_\alpha}[x_2, x_3, x_1, \ldots, x_{n+3}], \neg F_{aux4}[x_2, x_3, sk_\alpha(x_1), \ldots, x_{n+3}]\} \tag{19}$$

$$\{F_{aux4}[x_1, sk_G, x_2 \ldots, x_{n+2}], \neg F_{aux6}[x_1, x_2 \ldots, x_{n+2}]\} \tag{20}$$

Finally, to perform the fourth task, we add the following clauses:

$$\{F_{aux6}[x_1, \ldots, x_{n+1}, sk_\beta], \neg P_\beta[x_1, \ldots, x_{n+1}]\} \tag{21}$$

3.2.4. Mimicking the evolution of STMs in deterministic ATP

Applying the rules of our translation procedure to a given STM results in a set of Krom–Horn clauses. The initial and final states are translated into clauses of length 1, and all instructions are translated into clauses of length 2 with exactly one negated literal. Our presumed complete ATP procedure based on tight connection tableau, started with clauses that contain only negated literals, makes it possible to mimic the execution of deterministic

STMs via a very simple deterministic ATP process. It starts with the initial clause, since this is the only clause that contains only negated literals. From then on, there is only exactly one expansion step to be iteratively performed. Reduction steps, which do occur in a complete tableau proof search for the whole realm of FOL, never occur in the complete tableau proof search for the special case of Krom-Horn clauses. The proof search terminates if and only if the halting state is reached. Since the proof procedure is deterministic, any proof is necessarily of minimal length. Incidentally, the same is true for a proof search within the linear resolution calculus, which does not differ significantly from the proof search in tableaux in the very special case of Krom–Horn clauses.[9]

Although ATP mimics the evolution of STMs, there is no one-to-one correspondence but rather a one-to-many correspondence between the steps of the STMs and the steps in our tableau calculus. The main reason for this is Rule R, which implies a pop–push process for the Skolem functions in positions two and three of the predicates to obtain the correct order of the arguments in the Skolem functions. However, one can compare the literals in the ATP process after each translation of an instruction with the tape after the performance of an instruction during STM execution. This can be seen, for example, by comparing the color diagrams of the STM sequences with those of the ATP process; see, e.g., Figure 4 and Figure 5.

The translation T(STM1) of STM1, which is a provable formula, differs by only one literal in one clause from the translation T(STM2) of STM2, which is a satisfiable formula. The ATP proof of T(STM1) in tableaux takes 634 steps. Figure 5 shows the evolution diagrams for T(STM1) and T(STM2), divided into three parts (the partial diagrams are to be read from top to bottom). The outermost left partial diagram shows steps 1–96, and the next partial diagram presents steps 97–346 after the first splitting. The third partial diagram shows steps 347–634 for T(STM1) after the second splitting, while the last partial diagram depicts steps 347–640 for T(STM2) after the second splitting. The third and fourth partial diagrams deviate only in step 634, which is the last step in the proof of T(STM1), while the proof search for T(STM2) goes on forever. The patterns repeat after each splitting, becoming more nested; this extends the pop–push processes without causing new symbols to be written or new states to be entered relative to the previous sequence of inference steps within one splitting period. Thus, similar to the computable sequences of STM1 and STM2, the sequences of inference steps for T(STM1) and T(STM2) share the same repeating pattern. However, this pattern repeats endlessly only in the proof search for T(STM2). This demonstrates that the invalidity of the loop criterion for deciding whether halting occurs in the case of STM1 and STM2 extends to the invalidity of deciding provability for T(STM1) and T(STM2) based on a loop criterion for an automated proof search within a tableau or resolution calculus.[10]

Since we can produce an arbitrary number of further pairs of STMs for this case, we can also generate an arbitrary number of further pairs of provable and satisfiable formulas that share a regular sequence of inference steps up to an arbitrary length. This demonstrates that specifying a loop criterion for ATP for clauses with Skolemization is futile without further restrictions to circumvent such cases.

3.3. *Proof search without Skolemization*

Decidability, if considered from a computational, intensional point of view, may well depend on notation. Skolem functions allow for nesting and, thus, for representing the splitting of cells on the tape. However, one can eliminate Skolemization while preserving satisfiability. Therefore, the question arises of whether the problem of specifying a loop

[9] Full resolution reduces to binary resolution, factorization is not needed, the expansion rule and binary resolution are identical in this case, and [7] have shown that linear *input* resolution is complete for Horn clauses.

[10] T(STM1) and T(STM2) each contain 79 clauses and thus are too long to be printed here. All automatically generated diagrams and translations from STM1 and STM2 can be viewed here . The printed output of the color diagrams and their symbolic expressions is 125 and 133 pages long.

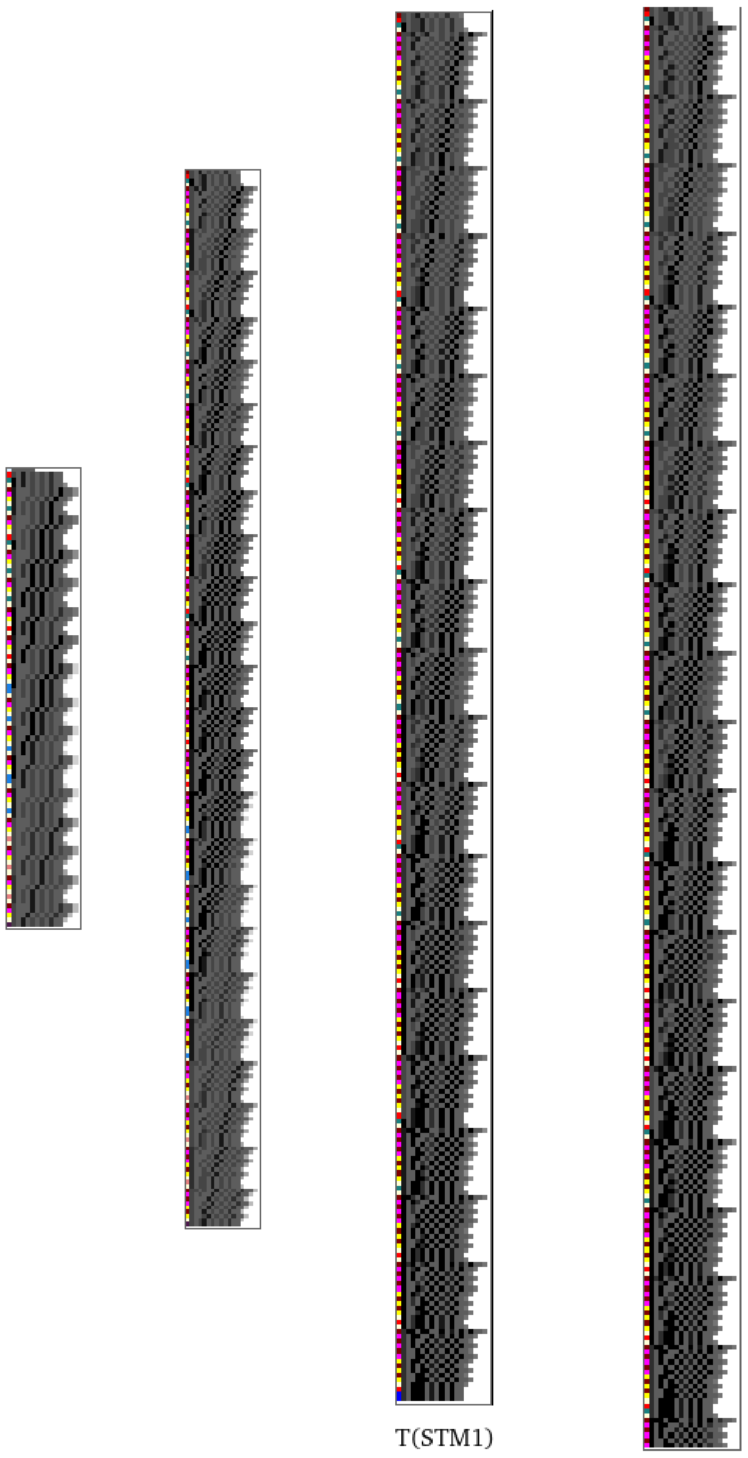

Figure 5. Color diagrams of the proof searches for T(STM1) and T(STM2), showing that although both are governed by the same pattern before the proof search for T(STM1) terminates, only the proof search for T(STM2) continues to repeat this pattern.

criterion can be overcome by referring to a translation into pure FOL and a calculus that is based on nothing but equivalence transformation within pure FOL.[11]

To investigate this, we specify a procedure for translating the tableau proof search into formulas for a proof search within a calculus in which $A \vdash A \wedge A$ (= $\wedge$I) is applied as the only rule that iteratively increases complexity. We formulate a complete calculus based on this rule and its iterative application to anti-prenex and, thus, negated normal forms (NNF). We call this calculus the "NNF calculus". It starts from anti-prenex normal forms that cannot be decided based on trivial criteria, to the effect that no $\wedge$I applications are needed. Similar to the iterative utilization of clauses in tableaux or resolution, the proof search in the NNF calculus then consists of computing iterative applications of $\wedge$I to multiply universal expressions, to the effect that if a proof is found, then the resultant expression can be transformed into a prenex normal form such that all substitutions needed to refute the resultant formula can be achieved via universal quantifier elimination.

Our STMs are designed not only for a simple and direct translation into clause forms and their proofs but also for the translation of these clauses and the tableau proof search into formulas of pure FOL and their proofs within the NNF calculus. The application of $\wedge$I can be encoded in a formula that entails the intended substitutions to unify pairs of literals that correspond one to one to the unification of the corresponding literals in the tableau proof search. Our translation of the tableau proof search into a proof search in the NNF calculus yields a proof in the NNF calculus if and only if one is found in the tableau calculus. Since our translation procedure is based on a deterministic proof search in tableaux, this can be done without the need to apply the cumbersome procedure of [2], section 19.4, to eliminate function symbols. We skip the details here; cf. our commented implementation for details (cf. section 4).

The important differences in syntax between the tableau proof and its translation into the NNF calculus are not only the lack of Skolem functions but also the fact that the tableau proof search utilizes clauses of length 2 (or less), while the corresponding NNF proof search relies on applications of $\wedge$I to universal expressions that differ in length. The length of these universal expressions differs with the length of their final scope, which depends on the initial scope of the universal expressions and on the selected literals from this scope that are utilized to be unified. These differences lead to a significant difference in the proof search patterns. This can be seen in the color diagrams of the deterministic ATP processes for Krom–Horn clauses and their translations. Whereas it is sufficient to depict a sequence of expansion steps on the main branch in tableaux, we cannot represent the sequence of $\wedge$I applications by only one sequence of color diagrams. This is so because of the existence of $\wedge$I applications to universally quantified expressions with existential quantifiers above conjunctions in their scope; these conjunctions, in turn, have universal quantifiers above disjunctions in their scope. To these universally quantified expressions, $\wedge$I can, in turn, be applied. Therefore, the color diagrams need to represent sequences of $\wedge$I applications to the former universally quantified expressions ($\forall\exists$ expressions for short) as well as to the latter ($\forall\vee$ expressions for short) within these expressions; cf. Figure 6.

We indeed find that certain problems that arise with the loop criterion in tableaux do not arise within the NNF calculus. However, the principal problem with a loop criterion that we find for T(STMs) also arises for their translations TT(STMs) into pure FOL formulas and their proof searches in the NNF calculus: the proof searches in the cases of a provable formula and an unprovable (satisfiable) formula may go hand in hand in accordance with a repeating pattern until a proof is found for the provable formula, while the search goes on forever in the case of the unprovable formula.

This is proven by our translations of T(STM1) and T(STM2) into pure FOL formulas, TT(STM1) and TT(STM2), and the translation of the corresponding tableau proof into a final expression encoding the corresponding proof search in the NNF calculus (cf. Figure 6; for details, consult the link given in footnote 10). The diagram of the NNF proof search

[11] In fact, [11], section 6, argued as much. The present paper explains why this attempt was mistaken.

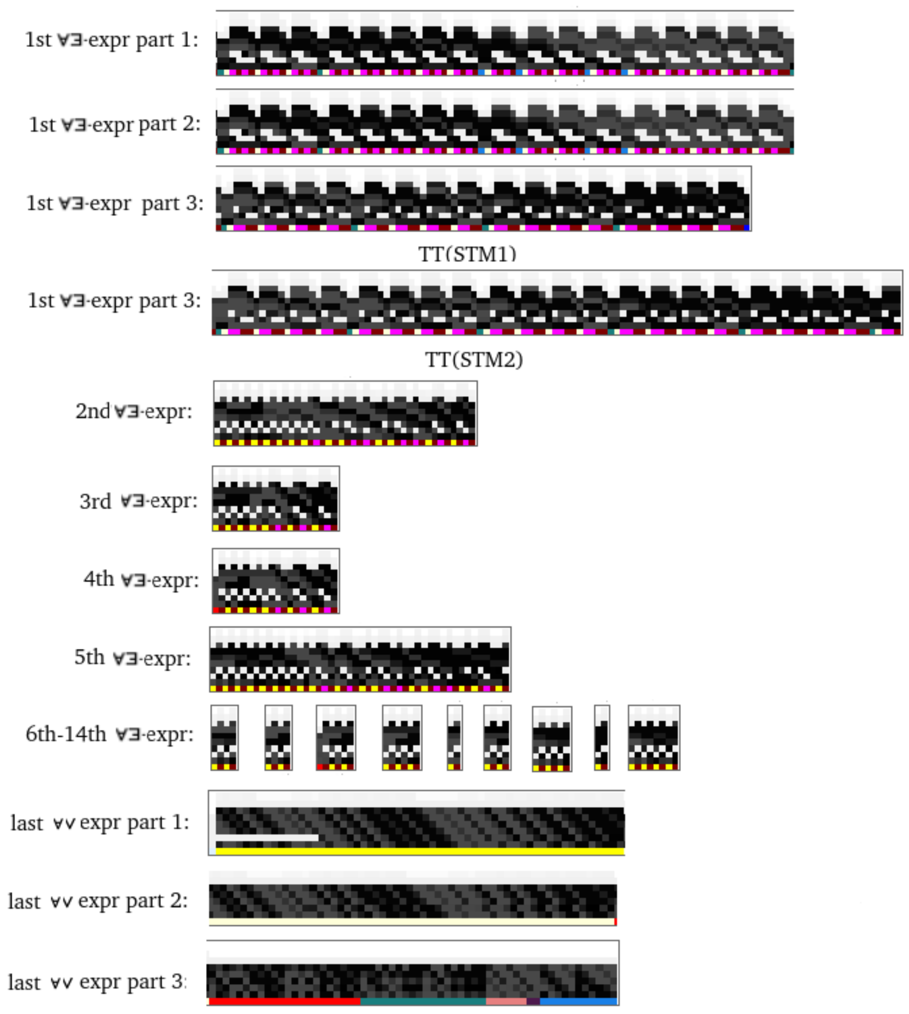

Figure 6. Color diagrams of the NNF proof searches for TT(STM1) and TT(STM2), showing that although both are governed by the same pattern before the proof search for TT(STM1) terminates, only the proof search for TT(STM2) continues to repeat this pattern.

for TT(STM1) and TT(STM2) corresponding to the first 634/640 steps of the tableau proof search for T(STM1) and T(STM2) consists of 15 partial diagrams. The first 14 depict a sequence of multiplied ∀∃ expressions, all stemming from multiplying the same initial expression by different literals selected from its scope, while the last diagram contains ∀∨ expressions not occurring in ∀∃ expressions. The ∀∃ expressions may, in turn, contain multiplications of ∀∨ expressions. Therefore, the diagrams depicting ∀∃ expressions may, like the last diagram, contain diagrams depicting sequences of ∀∨ expressions Only the first partial diagram contains the significant difference between the proof of TT(STM1) and the endless proof search for TT(STM2). We split this partial diagram into three parts. They show a repeating pattern until the diagrams in Figure 6 for TT(STM1) and TT(STM2) differ, in the last column of "1st ∀∃-expr part 3" for TT(STM1) and the corresponding column of "1st ∀∃-expr part 4" for TT(STM2), which depicts only one inference step in an endless proof search. This corresponds to the difference at step 634 in the tableau proof search for T(STM1) and T(STM2). The remaining 14 color diagrams in Figure 6 are identical.

4. Materials and Methods

In contrast to metamathematical proof methods, including the diagonal method, we base our results on examples and computation. STMs and their automated translations into Krom–Horn clauses and pure FOL formulas are especially suited for explaining undecidability in terms of pattern detection in ATP. To illustrate this as simply as possible, we compute color diagrams instead of printing complex symbolic expressions and proofs.

Our results are based on a *Mathematica* program called *KromHornSolver* that we implemented ourselves. This program takes an STM plus an upper bound for execution as its input. It then performs the following steps:

Step 1: Generate the array for the steps of the STM and print its color diagram.

Step 2: Translate the STM into clauses (with Skolem functions) and into a pure FOL formula (without Skolem functions).

Step 3: Generate the deterministic tableau for the clauses and print its color diagram.

Step 4: Translate the tableau proof into a pure FOL formula encoding the corresponding proof in the NNF calculus and print its color diagram.

Step 5: Check attempts to specify a loop criterion.

This program is available here, and an implementation of it can be accessed here.

In the final step, our procedure automatically checks attempts to specify a loop criterion. Prior to the implementation of the *KromHornSolver*, we were unable to generate a counterexample for the implementation of a loop criterion within an automated proof search in the NNF calculus.

5. Discussion

Our intention is to address questions of (un)decidability in the context of the concrete everyday struggle of a software engineer to solve problems to a maximal extent by computation. We believe that this kind of endeavor significantly enhances our understanding of the limits of decidability and our ability to explain these limits.

Our main concern is to *demonstrate that* and to *explain why the particular putative attempt* to solve the *Entscheidungsproblem* within *given* calculi and proof search algorithms by adding a general loop criterion is futile instead of *proving in general that* the *Entscheidungsproblem* is unsolvable *by any method*. Our explanation is based on mimicking STM sequences in ATP. However, this does not mean that our explanation depends on the translation of STMs into FOL. We merely use this translation method as a heuristic to generate pairs of sequences in ATP that share a repeating pattern. As soon as one has generated those sequences, the validity of a loop criterion for ATP is directly refuted by the fact that an infinite sequence for an unprovable formula and a finite sequence for a provable formula contain the same repeating pattern.

One can regard our discussion of the limits of a loop criterion as an instance of the general insight that finite, regular parts of computable sequences do not have an unambiguous continuation. However, we believe that it is not sufficient to refer to this insight in general in the context of ATP. Instead, one must prove that and how such a general statement indeed applies to the problem of specifying decision criteria within ATP via pattern detection, which we do by means of the *KromHornSolver* and its application to cases such as the translations of STM1 and STM2 in our example. We do not claim that our explanation generalizes to *any* ATP without further ado. Instead, on our view, merits and limits of specific attempts to decide FOL are in need of being investigated individually.

Further questions arise in regard to a loop criterion. One can ask how a loop criterion relates to the fact that there exist computable sequences of arbitrary length that do not contain any partial repeating sequences, so-called *infinite square-free words*. The repetition in question in square-free word sequences concerns regularity in the *narrow* sense. One might ask whether this can be generalized to regularity in the *broad* sense and how this applies to ATP. This concerns the question of the extent to which versions of a loop criterion can ensure termination.

On the other hand, specifying the limits of a certain attempt to define a decision procedure may also yield a discussion of the merits of that attempt if restricted to partial solutions for a decision problem. In this paper, we have only considered the limits of a loop criterion in ATP, without discussing its possible merits. However, in our view, the discussion of the limits of decidability should be twofold and, thus, also explain and specify to what extent a decision problem can be decided based on a proposed criterion. We have experimented with many different versions of a loop criterion in ATP. There is always the danger of generalizing from regular sequences of proof searches in the case of satisfiable formulas such that certain proofs of provable formulas are no longer captured. For example, the pop–push process that is part of the translation of Rule R makes evident that inference steps may be repeated to a great extent before some difference arises that enables significant progress in seeking a proof. However, we still believe that a restricted version of a loop criterion would be a highly significant contribution to the method of saturation in ATP. Most interestingly, the success of a restricted loop criterion strongly depends on the syntax to which it is applied. We intend to contribute to this topic in our further research.

Author Contributions: Both authors contributed equally to this work and have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Pontifical Catholic University of São Paulo, grant number 21466, under the Internationalization Incentive Program (PIPRINT EDICT 11802/2021). The responsibility for the content of this work and its respective dissemination/publication is exclusive to the authors. The opinions, statements, conclusions, and recommendations expressed in this paper do not necessarily reflect the views of the grantor on this subject.

Data Availability Statement: Our program *KromHornSolver* is available here, and an implementation of it can be accessed here.

Acknowledgments: We are grateful to Victor Rodych, Markus Säbel, Karsten Müller and David Fuenmayor for comments on this paper.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analysis, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ATP	Automated Theorem Proving
DNF	Disjunctive Normal Form
FOL	First-Order Logic
NNF	Negated Normal Form
STM	Splitting Turing Machine

References

1. Börger, E.; Grädel, E.; Gurevich, Y. *The Classical Decision Problem*, 1st ed.; Springer: Berlin, 2001.
2. Boolos, G.S.; Burgess, J. P.; Jeffrey, R. C. *Computability and Logic*, 4th ed.; Cambridge University Press: Cambridge, 2003.
3. Dreben, B.; Goldfarb, W.D. *The Decision Problem. Solvable Classes of Quantificational Formulas*; London: Addison-Wesley: London, 1970.
4. Gödel, K. On Formally Undecidable Propositions of Principia Mathematica and Related Systems I. In *Kurt Gödel: Collected Works*; Feferman, S. et al., Eds.; Heijenoort, J.v., Transl.; Oxford University Press: New York, US, 1986; 145–195.
5. Floyd, J. Wittgenstein's Diagonal Argument: A Variation on Cantor and Turing. In *Epistemology versus Ontology, Logic, Epistemology*; Dybjer, P., Lindström, S., Palmgren, E., Sundholm, G., Eds.; Springer: Dordrecht, Netherlands, 2012; 25-44.
6. Floyd, J. Turing on 'Common Sense': Cambridge Resonances. In *Philosophical Explorations of the Legacy of Alan Turing*; Floyd, J., Bokulich, A., Eds.; Springer: Boston, US, 2017; 103–149.
7. Henschen, L., Wos, L. Unit Refutations and Horn Sets, *Journal of the ACM* 21.4, 590—605.
8. Kripke, S. Gödel's Theorem and Direct Self-Reference. *The Review of Symbolic Logic* **2021**, online first DOI:10.1017/S1755020321000526, 1–6.
9. Lampert, T. A Decision Procedure for Herbrand Formulae without Skolemization, Available online: https://arxiv.org/abs/1709.00191 (accessed on 20th December 2021), 1–30.
10. Lampert, T. Iconic Logic and Ideal Diagrams: The Wittgensteinian Approach. In *Diagrammatic Representation and Inference. Diagrams 2018*; Chapman, P., Stapleton, G, A. Moktefi, A., Perez-Kriz, S., Bellucci, F., Eds.; Springer: Switzerland, 2018; 624–639.
11. Lampert, T. Decidability and Notation. *Logique et Analyse* **2020**, *251*, 365–386.
12. Lampert, T., Nakano, A. Deciding Simple Infinity Axiom Sets with One Binary Relation by Means of Superpostulates. In *Automated Reasoning. IJCAR 2020*; Peltier, A., Sofronie-Stokkermans, V., Eds.; Springer: Switzerland, 2020; 201–218.
13. Letz, R., Stenz, G. Model Elimination and Connection Tableau Procedures. In *Handbook of Automated Reasoning 2*; Robinson, A., Voronkov, A., Eds; Cambridge University Press: Cambridge, US, 2006; 2015–2113. .
14. Neary, T. Woods, D. Four small universal Turing machines. *Fundamenta Informaticae* **2009**, *91*, 105—126.
15. Salehi, S. On the Diagonal Lemma of Gödel and Carnap. *Bulletin of Symbolic Logic* **2020**, *26.1*, 80–88.
16. Turing, A. On Computable numbers with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* **1936/37**, *2(42)*, 230–265.
17. Wolfram, S. *A New Kind of Science*; Wolfram Media: Canada, 2002.