Holography For Satisfiability I

Adewale Oluwasanmi
Independent Researcher
waleoluwasanmi@yahoo.com

Abstract

We present a meta-computational logic for Boolean Satisfiability that generalizes Holographic Algorithms to higher-dimensional *logical* manifolds. This logic is used to reason about both satisfiability and unsatisfiability proofs, *as functions*, which can be integrated and differentiated. We introduce twisted rings of proof integro-differential operators and a dualized algebra over these rings — along with an *algorithmic* ultraproduct construction which unifies an arbitrary number of such rings into a single, discrete topological ring, under bounded witness depth. The resulting framework expresses an algorithmic proof system, which derives proofs, surprisingly, in polynomial-time, with respect to the input size of a 3-SAT formula. As a by-result of these constructions, our framework offers a new descriptive complexity lens on the conflated class, *NP* ∪ *CoNP*, which is characterized by an algebraized Second-Order Logic with a Relative Least Fixed Point [A-SO-RLFP] that, in its implications, challenges the Church-Turing thesis, and our current understanding of the class of computable functions.

Keywords— Holographic Algorithms, Boolean Satisfiability Proof System, Integro-Differential Logic, Derived Algebraic Geometry, Algorithmic Reasoning, Algebraic Second-Order Computational Logic.

Contents

1 Introduction

1.1 Holographic Algorithms

In [1], Valiant introduces us to the concept of Holographic Algorithms. Informally, they can be described as algorithms that operate on a Holographic Reduction as a data structure. A holographic reduction is defined as a constant-time reduction that maps solution fragments many-to-many such that the sum of the solution fragments remains unchanged.

The reductions used by a holographic algorithm are (supposed to be) efficiently computed by polynomial equations that are defined as *gadgets*, which implement the *steps* of the algorithm. The computed reductions, called the Holant (omitting for now the distinction between generators and recognizers as used by Valiant), computed for some problem encoded within a graph structure G, can be expressed succinctly as:

$$\text{Holant}(G) \;=\; \sum_{\sigma:V\to\{0,1\}} \prod_{e\in E} f_e\big(\sigma|_{V(e)}\big). \tag{1}$$

Where V is the set of vertices (representing variables), E is the set of edges (representing constraints), $\sigma : V \to \{0, 1\}$ is a boolean assignment to all vertices, $V(e)$ denotes the set of vertices incident to edge e, $\sigma|_{V(e)}$ is the restriction of σ to those vertices and f_e is the constraint function associated with edge e and is realized as the equality (we use equivalence in our work) function on each v in $V(e)$.

As such, the expression is a sum over all variable assignments, represented by vertices $v \in V$, the product of every constraint on each v. Note that edges may also carry varying weights, resulting in a weighted sum.

The complete, basic holographic framework consists of *generators* which emit solutions in combinations along the *gadgets*, and *recognizers* which absorb these combinations (and may further re-emit them). A much more extended treatment that formalizes generators/recognizers can be found in the work of Cai et al [2], which we utilize here in an extended way, as a guiding framework.

1.2 The 3-SAT Problem

The 3-Satisfiability problem is a well-known problem in the field of computer science and mathematical logic, notable for being one of the quintessential problems in the *NP-Complete* space. The full classification of the space of *NP* problems as well as the question of whether $P = NP$, had the earliest developments in the combined works of Cook [3], Levin [4] and Karp [5]. In their *normal* form, satisfiability clauses are expressed as bracketed disjunctions $\vee$, over a set of subscripted variables $x_1, \ldots, x_n$. For example:

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee .... \vee x_n) \textit{ up to any number of } n \textit{ variables.}$$

Satisfiability formulas join these clauses by conjunction, $\wedge$, and allow variables x_i and their negations $\neg x_i$ to be repeated in separate clauses, up to any number of clauses. For example:

$$(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4 \vee .... \vee \neg x_n) \wedge$$
$$(\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee .... \vee x_n) \wedge ... \wedge$$
$$(\neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee .... \vee \neg x_n) \wedge$$
$$(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee .... \vee x_n).$$

Satisfiability problems in this form then ask for a satisfying model, in this case, a set of assignments to the unit variables in the formula's clauses that make all the enclosed clauses evaluate to *true*. The 3-SAT variant of the satisfiability problem, which is the variant we address in this work, places the extra limitation that no clause contain a bracketed disjunction of more than 3 variables.

1.3 The Main Result

The main result of this work shows that:

1.3.1 Holographic 3-Satisfiability

Determining if a 3-SAT formula F, is satisfiable, is equivalent to computing an extended type of Holographic Reduction R for the solution space of F, and finding a satisfying solution to F is equivalent to *collapsing* R into a single solution/set of assignments S. Thus, reductions are effectively encoded as (higher dimensional) logical models in which 3-SAT formulas can be interpreted.

1.3.2 Integrable Reductions

Such holographic reductions can be *integrally* computed. The gadget for this becomes an integral operator.

1.3.3 Differentiating Gadgets

Differential operators, also realized as gadgets, can operate over integrated reductions to yield the needed collapse to a single solution.

1.3.4 Algebraizable Calculus

Both sets of gadgets, as operators, can be encoded as elements of algebraic structures other than (polynomial) equations. These elements can then be manipulated as *values* by an algebra, where the process of realizing this algebra for any formula F, is an *algebraization*. The underlying calculus, along with the operators, is thus, algebraized.

1.3.5 Abstract Algebraic Logic

Since solutions computed by gadgets are *proofs* (*UNSAT* certificates or satisfying solutions to a *satisfaction* problem), the algebraized calculus *realizes* a logic — a first-order logic. The algebraization also defines second-ordered operators, as well as a *least fixed-point*, which *relatively* bounds the second-ordered operations so that we obtain *bounded operators*. The associated *relativity* derives from the fact that the fixed-point is determined by (the unique clausal structure of) the input formula. The algebraization process, Γ, is well-defined as a set of operations that reason about this first- and second-order elements as algebraic structures, and thus, is an *abstract algebraic* logic.

1.4 Algebraization Strategy

Γ, as we realize it, is computational, meaning that it is analogous to *computer algebraization* — the conversion of algebraic structures into compatible algorithms and data structures. In our case, these structures originate from an algorithmic context and so, come, already, as a matter of fact, converted. Thus, their *algebraic* properties originate from, and are tied to, the computational structures that encode them.

Reductions are presented via structures which encode point-like *algebro-geometric* objects that live on differentiable manifolds while operators take the form of structures which encode *differential algebraic* objects that operate, in a compatible manner, on points of this manifold. Both forms of encoding are mapped *implicitly* by our algorithmic realization, to the same underlying set. Reductions and operators as *first-ordered* algorithmic objects, along with their second-order bounds, encoded as set limits, are then computed by an algebraizing algorithm $\mathcal{P}$ which implements Γ. $\mathcal{P}$ functions as a meta-algorithm, in the sense that the instance-specific algebras over the ordered objects constitute the (regular, holographic) algorithmic space. $\mathcal{P}$ is presented as a *Second-Order(ing)* Proof Calculus.

2 Overview Of The Calculus

As noted in the introduction above, our proof calculus functions at two levels of logical ordering. We first briefly describe the first-order structure, which corresponds to the *usual* proof calculi. We then describe a second-ordered operator-valued level via the meta-model structure it defines.

2.1 Logic As Calculus

In normal, that is, traditional use, the term, Proof Calculus *PC*, in mathematical logic, refers to a system for inferring theorems of some language of formulas, L , given some starting set of axioms, A, and some rules of inference, R. The prototypical calculus can be represented by the following tuple:

$$PC = (L,\ A,\ R)$$

Different types of Proof Calculi place emphasis on the importance of axioms (Hilbert Systems) versus Rules Of Inference (Gentzen-style systems). There exist other types of Proof Calculi, but for our purposes, these two are sufficient for representing the general classifications of Proof Calculi. Our proof calculus, at the first-order level, follows the general recipe for a Proof Calculus outlined above, but in a way that mimics Pure Calculus, so that:

- L consists of functions.
- A consists of the definitions and invariants of differentiation and integration and the fundamental theorem(s) dualizing both operators. These definitions and invariants are in all cases, realized by functions, that is, algebraically defined differential and integral operators, of these operations (differentiation and integration).
- R consists of rules of operation.

We explore each element of the tuple in more detail:

- By integrating and differentiating proofs, the calculus manipulates ***proofs as functions***. This variation on the normal proofs-as-programs concept implies that the operators of the calculus function as *logical* operators over proofs, therefore a more technically accurate term for this level of the calculus would be a ***Calculus Of Proofs*** or a ***Proof-Valued Calculus***. Proofs-as-functions, thus, constitute L.
- The rules R, of the calculus, outline the concrete steps of differentiation and integration, as they would be carried out on instances of functions. Elements of R state *what an operator* ***does***.

- Definitions, which state *what an operation* ***is***, as well as operational invariants, which comprise of technical conditions that each operator must preserve at each point in time, together comprise A. As stated, these definitions and invariants are concretely realized as properties of algebraically defined differential and integral operators.

As such, our proof system differs in character from many standard proof calculi and systems by not relying directly on formal propositional logic but instead, building up a concept of proof calculability from the ground up. We achieve this by drawing inspiration from these traditional proof frameworks, as well as by borrowing concepts and constructions from a variety of mathematical and physical disciplines, as will be made clear in the text. To make the content accessible to a reader with a background in computer science theory and/or logic, we subsume all these different influences under the overarching rubric of an algebraic and *algorithmizable* computational logic. The algorithm *recovered* from this process is a *holographic algorithm*.

2.2 The Reverse Mathematics Of Computation

Our strategy for obtaining the full theory of the calculus utilizes a variant of the *reverse mathematics* program, which seeks to determine the set of necessary axioms needed to prove theorems of mathematical domains. This program, as practiced in the literature, works in ***reverse***, by starting with theorems and then fleshing out the axioms of the domain, necessary for proving the theorems, in contrast with normal mathematical practice.

Our general analysis will follow this scheme, fitted to a computational context, for describing the calculus. We frame this computational variant as *the reverse mathematics of* ***computation***. Our extended scheme adheres to the following general principles:

Algorithms First Starts with concrete computational steps which delineate the algorithms that define the operations. From these, we derive the rules R of the system.

Functions And Theorems We then identify the types of functions computed by these algorithms, along with their domains, and then prove theorems about these functions. These identify the *language* L.

Definitions/Properties Using these theorems, we determine the definitions imposed by the theorems on the structure of the operators as well as any properties of the operations that need to be preserved before, and after, each step. This establishes the *axiomatic* basis A.

2.3 Algebro-Geometric Meta-Model

In addition, we subject the calculus, if indeed it may be described as such, to a condition that expresses the fact that the operators must be dualized, over every possible function, implying that the calculus be sound and complete over the space of proofs. To achieve this, we constrain and construct the model of the calculus in accordance with Tarski's undefinability theorem, which states briefly, that for any sufficiently complex formal language L, a truth predicate T for L cannot be defined within L but must necessarily be defined by a metalanguage ML. In essence, for our calculus to be sound and complete as a system which constructs logical statements, as judgements, about satisfiability, the *truth* of its model M (holographic reductions) must be guaranteed by a metamodel M^M so that if f_i is an interpretation in M and F_i an interpretation in M^M, then:

$$\forall i \in \mathbb{N}, f_i \iff F_i \text{ s.t. } f_i \equiv F_i.$$

The measure of adequacy of M^M as a meta-model will correspond to its capacity to produce 2 artifacts — a theorem analogous to the fundamental theorem of calculus, which dualizes the operations of the 2 operators — and a theorem which establishes its soundness and completeness for any 3-SAT formula. The dualization must function as follows : if L consists of functions (as proofs), then M^M must make inferences about the operators as proofs about proofs and must formally express a natural duality between *computing* the meta-proof (integration) of a proof, and *extracting* (differentiating) the proof from the meta-proof, in order for the meta-proof to be a meta-proof.

Mathematically, M^M is realized as the *topological boundary* of M, over some (differentiable) manifold *Manifold(M)*, equipped with (a *sheaf* of) logical constraints, in which M^M is used to ***reason about***, as well as ***index***, via operators, the interpretations of M, as well as their higher dimensional multiplicities, as will be demonstrated. M^M must then necessarily provide a formal *topological closure* for the interpretations of M as points on *Manifold(M)*. This closure will function as the second-ordered, complementary (outer measure of the inner) *relative least fixed-point*, of the total logical stack. M^M is thus expressed, as a second-order, *operator-valued* logic. For the reader's convenience, we include a *simplified* definition of a topological closure:

Definition 2.1. *The* topological closure *TClosure of a subset S of points in a topological space T is* **defined** *as the* union *of all points in S together with all limit points (boundary) of S. TClosure is further* indexed *by the intersection of all closed sets C (sets of points which also include their boundaries) in T, where* $S \subset C$.

3 The Basic Operations And Definitions Of The Calculus

We now flesh out the basic structure of the calculus, its operations, and definitions, as well as its motivating theorems.

Remark 3.1. *In this section, the operational space described for the solution space of a single clause/two clauses, may appear at first glance, to be discrete rather than continuous even though we do define a topology over it. The reader must recall that the intention is to use these fundamental spaces as base cases for induction of a topology on the solution space of a formula F with an arbitrarily large number of clauses and variables. The larger topology, as will be evident, is what we will refer to as an algorithmic ultraproduct construction from the topology defined on the solution space of individual clauses. To define this non-standard ultraproduct construction, for the standard (first-order) ultraproduct construction:*

$$\left(\prod_{i \in I} \mathcal{M}_i\right) / \mathcal{U}$$

We make the following adjustments:

- *Each* $\mathcal{M}_i$ *is one of the following:*
  - *The topology of the solution space of a clause, which we term, of type* C^n*, of some* 3*-SAT formula* F_n *with n variables.*
  - *The topology of the solution space of a clause, which we term, of type* C^j*, ultimately derived (via a bounded chain of derivations) from some* C^n *by the application of an ultraproduct rule (which we will define later as bounded witness correlation).*
  - *The topology of the solution space of a formula* $\bigwedge_{l=1}^{m} C_l, m \leq n, C_l \in \{C^n\} \vee C_l \in \{C^j\}$.
- *Each* $\mathcal{M}_i$ *is computed as a second-order (not first-order) structure.*
- *The finite ultraproduct* $\mathcal{M}$ *is the* bounded *topology of* $\bigwedge_{l=1}^{n} C_l$*, and is a second-order structure. The* boundary *of* $\mathcal{M}$ *is the operator* $\prod$.
- $\mathcal{U}$ *(the ultrafilter) is an* **algorithm** *that assigns truth assignments to all n variables so that* F_n *is satisfied – the satisfaction of* F_n *functions as a second-order substitution for Los's theorem.*

It is then straightforward to deduce, from this (second-order) ultraproduct expansion, that the equivalence relations obtained via an ultrafiltration over any one set of truth assignments to F, indeed correspond to continuous functions and rates of change over some finite subset of an (potentially) infinite set, as we demonstrate.[1]

[1] It is also a working fact in the mathematical community that (first-order) ultraproducts are a bridge between discrete and continuous spaces, for example, as demonstrated in the correspondence between discrete and continuous metric spaces such as the one we construct in this article.

3.1 Algorithmic Determinacy

A non-deterministic algorithm is usually defined as an algorithm that, even for the same input, can exhibit different behaviors, that is, produce different outputs, on different runs. This contrasts with a deterministic algorithm, which always produces the same output for a given input and follows the same execution path. This provides the fundamental motivation of the traditional definition of the *NP* class of problems via the behavior of its associated Turing Machine. For our purposes here, we give a more specialized definition of the notion of determinacy:

Definition 3.1. Deterministic Algorithm — *DetAlg: A deterministic algorithm is a* **one-way** *truth value assignment (*0 *or* 1*, with True* $\coloneqq 1$ *and False* $\coloneqq 0$*) sequence to a* satisfiable 3 *SAT formula* F_n *with* n *distinct variables :* $\{0,1\}^n \to \{0,1\}$*. This implies that DetAlg* does not *backtrack on any assignment.*

We now formalize a notation structure for assignments in a *DetAlg*.

Recall that variables in a 3-SAT formula are expressed as subscripted letters, where each subscript $i \in \mathbb{Z}^+$ indicates a distinct variable. In a *DetAlg*, we assume that only one letter is used to represent all variables, say, the letter x, and that distinct variables are identified by their subscripts (as described in the introductory section).

Convention 3.1. Assignment Notation*: Whenever we assign a variable with subscript* i *to the value* 0*, we will notate the assignment as* -i *with a minus sign in front of it. On the other hand, whenever we assign* i *to* 1*, we will notate the assignment as* i*, with no sign prepended.*[2]

Example 3.1. *In the clause* $(x_2 \vee \neg x_3 \vee \neg x_4)$*, if we assign* x_2 *to the value* true *(*1*), the assignment will be represented as* 2 *and if we assign it to* false *(*0*), the assignment will be represented as* -2*. Likewise, for* x_4*, if assigned to* 1*, the assignment will be represented as* 4 *and if assigned to* 0*, the assignment will be represented as* -4*.*

In a *DetAlg*, which assigns truth values to the variables of a satisfiable formula, (as well as anywhere else assignments are recorded in this article, except where noted), we will use Convention. 3.1.

Example 3.2. *Given the clause* $(x_1 \vee x_2 \vee x_3)$*, if we assign* x_1 *the value* false*, we will add/append the string '*-1*' to the representation (which we discuss next) of the DetAlg. On the other hand, if we assign* x_1 *the value (true), we will append the string '*1*' to the representation of the DetAlg.*

Definition 3.2. DetAlg Value Set — *DetAlgValSet: Any DetAlg can be represented by a singly-quoted, comma-delimited set of strings of assignments. We will denote such a set as its value set, DetAlgValSet.*

Example 3.3. *The DetAlgValSet '*$1, 2, -3, 4, 5, -6, -7, -8, 9, 10, -11, 12, 13, -14, 15$*' is a valid representation of a completed and satisfying DetAlg for a formula with* 15 *variables encoded within any number of clauses, as described in the introduction.*

Definition 3.3. Deterministic Sub-Algorithm*: A deterministic sub-algorithm B of some larger DetAlg A, is defined to be a DetAlg whose* $DetAlgValSet_B$ *is a strict subset of the* $DetAlgValSet_A$ *of A :* $DetAlgValSet_B \subset DetAlgValSet_A$

Example 3.4. *The DetAlg represented by the DetAlgValSet '*$1, 2, -6, -7, 9, 10, 13, -14$*' is a sub-algorithm of The DetAlg represented by the DetAlgValSet '*$1, 2, -3, 4, 5, -6, -7, -8, 9, 10, -11, 12, 13, -14, 15$*'.*

Definition 3.4. Point In Time*: A distinct point in time for a DetAlg D of a* 3*-SAT formula* F_n *with* n *variables, is a number* $t \in \mathbb{N}, 1 \leq t \leq n$*, where* t *denotes the* t^{th} *instant of truth assignment, in a sequence of truth assignments* $\mathcal{S}$*, to the variables in* F_n*.*

[2] In doing this, we follow the DIMACS notation format used in modern SAT solvers.

Definition 3.5. Time $t = 0$*: Define time* $t = 0$ *as a point in time when a DetAlg has not assigned a truth value to any variable of a* 3*-SAT formula.*

Definition 3.6. Free Assignment — *Free-Val: A free assignment* f*, is an assignment of a truth value* v*, to a variable* x_i*, by a DetAlg* D*, at a point in time* t*, such that some assignment* a*, made by* D *at an earlier point in time* $t-k, k>0, t-k>0$, does not force D *to assign* v *to* x_i *at time* t*:* $a \implies v \vee \neg v$ *— That this is indeed defined by an implication will be proved in Lemma. 9.1.*
This implies that D *could have assigned the opposite truth value* $\neg v$ *to* x_i *at time* t*.*

Definition 3.7. Bound Assignment — *Bnd-Val: A bound assignment* b*, is an assignment of a truth value* v*, to a variable* x_i*, by a DetAlg* D*, at a point in time* t*, such that some assignment* a*, made by* D *at an earlier point in time* $t-k, k>0, t-k>0$, forces D *to assign* v *to* x_i *at time* t*:* $a \implies v$*.*
This implies that D *could* not *have assigned the opposite truth value* $\neg v$ *to* x_i *at time* t*.*

We will describe, within the main body of the article, the process of exactly how truth assignments force other truth assignments in time.

3.2 The Problem Of First Ordered Determinacy

One can immediately infer that if we are to have correct (non-backtracking, satisfying) *DetAlg* instances, some structure must be doing the determination, which includes, forcing, of truth assignments to variables. This implies that valid instances of possible *DetAlg*s at any point in time, must be integrated, as well as interpreted, within some structure that orders the temporal sequencing of *DetAlg*s. We will **define** this structure as the *temporal model* of a *DetAlg*.

Recall that in Expr. 1, we identified a constraint on some vertex $v \in V$, as an equality function on v which maps to all of the variables (vertices) on the edges incident to v. We can equate this expression with a constraint of motion which expresses the fact, that from v, one can only *move* (while executing some deterministic algorithm), to vertices incident to v in the underlying graph structure. This First Order constraint can be stated as:
Let G be a graph, $\{v_1 \dots v_n\}$ be vertices in G
and e_{ij} be an edge in G connecting vertices v_i and v_j,
where $1 \leq i,\ j \leq n, i \neq j$, then:

$$\forall v, M(v_k, v_l) := \begin{cases} \top & \exists . e_{kl}. \\ \bot & \text{otherwise.} \end{cases} . \tag{2}$$

Where M is a *predicate of motion* that rejects all motions that are not scoped to a variable on an edge and accepts those that are. In other words, M orders the holographic algorithm in *time*.

Definition 3.8. First-Order Temporal Model — *TM-1 : We can redefine a Holographic Algorithm as a first-order structure that is ordered by Expr. 2 and for which Expr. 1 is a fixed-point, particularly a least fixed-point within a lattice ordered by transitions. This fixed-point is relative to the graph itself and so it is correct to* refer to *the underlying logic of the statement of Expr. 2 as a (holographic fragment of a) First-Order Logic with a Relative fixed-point, FO[RLFP].*

We ***define*** *the ordering:*

$$\textbf{\textit{assignment}}(a_i) \rightarrow \textbf{\textit{time}}(t_i), 1 \leq i \leq n, \textit{ for a formula } F \textit{ with } n \textit{ variables.}$$

that is, the mapping of truth assignments (to distinct variables) to distinct points in time, imposed by Expr. 2 — as a First-Order Temporal Model *TM-1. This model is denoted in use, simply as, TM-1.*

Ordering is **defined** in the context of *ordering by an algorithm*, and so we say a temporal model *orders a class of problems.*

Theorem 3.1. *$\exists\{F\}$ where $\{F\}$ is a set of 3-SAT formulas where each element F has a DetAlg that cannot be ordered by TM-1. And so, TM-1 cannot be used to (*generically/algorithmically*) order DetAlgs for 3-SAT formulas.*

Proof. We provide the following example of such an F.

Example 3.5. *Assume we are given a 3-SAT formula F_n with exactly 4 satisfying solutions and n variables. Assume F_n contains 2 variables, x_1 and x_2 along with other $n-2$ variables, which we will not distinctly identify:*

- *F has 4 DetAlg instances. Here we denote each DetAlg as $DetAlg_i$ where $1 \leq i \leq 4$.*
- *Further assume that each of the following $DetAlgValSet_i^2$s, of a sub-algorithm with just 2 assignments, holds in one, and only one, of the full $DetAlg_i$ instances, that is, the instance with a matching i value (note that each $DetAlgValSet_i^2$ is unique):*

  $$DetAlgValSet_1^2 = \text{`}1, 2\text{'}, \quad DetAlgValSet_2^2 = \text{`}1, -2\text{'}, \quad DetAlgValSet_3^2 = \text{`}-1, 2\text{'}, \quad DetAlgValSet_4^2 = \text{`}-1, -2\text{'}$$

- *Denote by $DetAlgValSet_i^n$, the complete DetAlgValSet of the full $DetAlg_i$ instance which contains all n variables.*
- *Now, further assume that the combination of truth assignments to the other variables (not x_1 or x_2) in each of the corresponding $DetAlgValSet_i^n$ instances is distinct/unique :*
  $DetAlgValSet_i^{(n-2)} \implies DetAlgValSet_i \ /\{x_1, x_2\}$,
  $DetAlgValSet_j^{(n-2)} \neq DetAlgValSet_k^{(n-2)},\ j \neq k$.
- *If at time $t = 0$, TM-1:*

  $$\boldsymbol{assignment}(x_1) \vee \boldsymbol{assignment}(x_2) \rightarrow \boldsymbol{time}(t_0)$$

  that is, ordering begins with an assignment of random truth values to either x_1 or x_2, then:

  $$((0 \vee 1) \rightarrow x_2) \in f_e(x_1)\big(\sigma|_{V(e)}\big) \quad \wedge \quad ((0 \vee 1) \rightarrow x_1) \in f_e(x_2)\big(\sigma|_{V(e)}\big)$$

  the constraint on the assigned vertex at that point in time would include both possible values for the other, as of yet, unassigned variable.
- *On the other hand, if TM-1:*

  $$\neg(\boldsymbol{assignment}(x_1) \vee \boldsymbol{assignment}(x_2)) \rightarrow \boldsymbol{time}(t_0) \ldots \boldsymbol{time}(t_{n-2})$$

  that is, ordering assigns truth values to every other variable in F_n before assigning a truth value to either x_1 or x_2 then,

  $$((0 \oplus 1) \rightarrow x_2) \in f_e(x_1)\big(\sigma|_{V(e)}\big) \quad \wedge \quad ((0 \oplus 1) \rightarrow x_1) \in f_e(x_2)\big(\sigma|_{V(e)}\big)$$

  the constraint on the assigned vertex at that point in time would include only one possible value for the other, as of yet, unassigned variable, depending on the particular instance.
- *This implies that TM-1 (the temporal model of FO[RLFP] with fixed-point as Expr. 1) is insufficient for and cannot be used to order DetAlgs for the class of satisfiable 3-SAT formulas.*

□

3.3 Exploring Second Ordered Determinacy

The situation in Example. 3.5 calls to mind again, Tarski's undefinability theorem.
We know that FO[RLFP] as we have introduced it, is sound for many problems, as shown by several implementations of holographic algorithms [1, 2, 7, 8, 9, 10, 11]. What unites the approaches in these implementations is the *graph-theoretic domain* of problems that they address. What we propose and go on to construct in this effort, is a method for reasoning about, and computing, both sound and *complete* models/reductions for holographic algorithms for **any** problem $M \subseteq NP$, in some other mathematical setting - which in this work is (logico-)algebro-geometric. The computed reduction should function implicitly as a predicate, as in Expr. 2, by filtering out unsatisfying/no longer satisfying solutions, for example, to a 3-SAT formula, at each step of a deterministic process (*DetAlg*). Ultimately, however, our constructions are inherently (Cayley) graph-like in the limit, and can be re-interpreted, if one is so inclined, as a type of *piecewise graph* theoretical framework.
We start with the following observations:

Observation 3.1. *Expr. 1 uses positive constraints to restrict algorithmic progression. That is, it only denotes, via functions defined over edges, vertices to which an algorithm can transition to, from any one particular vertex (positive forward motions).*

Observation 3.2. *The gadgets that implement those motions are defined by (polynomial) equations.*

Observation 3.3. *There are proof systems described in the literature that use equational models [12, 13, 14, 15], and several that use equational models in the form of a Nullstellensatz (primarily weak, and for refutations) proof procedure, as well as other related tautological systems for attacking propositionally (including Boolean Satisfiability) and other algebraically posed problems [16, 17, 18, 19]. There has also been use of Positivstellensatz approached [20]. In these frameworks, the problem is encoded into systems of polynomial equations or tautologies, and refutations proceed by polynomial derivations or some form of logical reasoning.*

Many of the systems mentioned have been established as having strong exponential lower bounds for both Nullstellensatz (NS) and Polynomial Calculus (PC) systems. Classic results in the field include superlinear or linear degree for algebraic proof systems [16] and propositional proof systems using Tseitin tautologies [20]. Notably, Alekhnovich and Razborov developed techniques to prove strong degree lower bounds for the Polynomial Calculus on families with expander clause–variable incidence graphs [19], this being comparable, with certain framework alterations, as noted below, with our approach.

We intend to address the points of blow-up identified in these systems, NS and PC, as well as propositional proof systems, w.r.t. Boolean Satisfiability, with our *polynomially* bounded, operator-valued logic, based on the following distinctions:

- **Different proof objects.** Proofs in NS/PC are sequences of polynomial manipulations, and in propositional systems, tautological arguments, whereas our full system reasons about *operator-valued* objects.
- **Different complexity measure.** In NS/PC the central parameter is *degree*. In our system, runtime and proof complexity are governed by the notion of a *witness boundary*.
- **Different operational semantics.** Our calculus uses holographic reductions and integro-differential operators, which have no straightforward simulation in low-degree polynomial derivations or tautological chain expansions. Incidence graph constructions as used in [19] are however, comparable (since as we have noted, our framework reduces to *incidence* graphs (geometrically) in the limit) if one introduces the notion of witnessable, bounded (by a second-order) computation.

We will use insights from these observations, primarily with regards to structuring a proof system, to first design and construct a small *second-ordered* predicative system. This means that we first (constructively) find the smallest 3-SAT formula instances for which the operators of the calculus can be well defined as well as temporally ordered. After the small predicative structure has been formally constructed and defined,

we will proceed to use inductive arguments/ultraproduct constructions, to build a larger predicative system that extends the structure of operations defined in the small system, as well as preserves their algebraic invariants.

3.4 Differentiation

In this section, we constructively define a differentiation procedure for the proofs of a 3-SAT formula with exactly 1 clause, that is, our small system, using the reverse mathematical scheme we outlined earlier.

3.4.1 Algorithmic Basis Of A Simple Derivation

Remark 3.2. *Many of the uses of differential terms in the next two sections may strike readers familiar with SAT solving techniques and/or Automated Theorem Proving as a replacement for certain terms as used in the existing practice and literature. The reader should remember that the uses in those contexts are not (fully) deterministic in the way we intend for the operations in this work to be. For the sake of clarity however, we will note any correspondences known to the authors after each applicable use of such a term.*

We start with Observation. 3.1, and from Theorem.3.1, conclude that capturing positive constraints is not sufficient for fully determining a holographic algorithm for a 3-SAT system of clauses. We then make an adjustment to our process of capturing constraints for a *single clause* by:
First substituting the graph structures used in traditional holographic algorithms with the notion of a more general solution space, θ, **defined as**:

Definition 3.9. Solution Space Of A Clause, θ *: Let* $v_1, \ldots, v_3$ *be the set of variables in a* 3*-SAT clause* C*. Then* θ *for* C *is the structure given by:*

- *The set of all satisfying solutions (sets of truth value assignments) to* C*,* A *:* $\{v_1 \coloneqq 0 \oplus 1, \ldots, v_3 \coloneqq 0 \oplus 1\}$ *(*$\oplus$ *denoting the XOR operation).*
- *And equipped with the operation of clause level assignment,* $[\coloneqq]^3 : [0,1]^7 \to [0,1]$.

Note that a single 3-SAT clause has $2^3 - 1$, that is, 7 satisfying solutions, which is the arity of the domain of the assignment operation.

Definition 3.10. Internal Nodes Of θ *: Let* C *be a* 3*-SAT clause, and:*

- S *be the set of all satisfying solutions* $\{A_k\}$ *to* C*,* A_k *:* $\{v_1 := 0 \oplus 1, \ldots, v_3 := 0 \oplus 1\}$.
- *And* $\int_i \subseteq S$.

Then, $\int_i$ *is an* internal node *of the solution space* θ *of* C.

From our conclusion above about the limitations of using only positive constraints, we can infer that we also need to define some function $\mathcal{F}$, equivalent to a product of constraints on a single clause, that denotes — with the term *denotation* as used in program semantics — that is captures in concrete terms, the domain of — the *internal nodes* of θ that we cannot move to at any point in time.

We refer to $\mathcal{F}$ as the negative (*negatively defined*) **constraint function**. For a small system, there is one clause, so there is exactly one such $\mathcal{F}$. We establish the definition of $\mathcal{F}$ with the following :

Definition 3.11. Unassignable Statement*:*

- *Let* $\mathcal{A}$ *be an algorithm that assigns truth values to variables of some clause* C.
- *And* ρ *be a statement which is a bracketed conjunction of truth assignments* $0 \oplus 1$ *to not-yet-assigned variables* $\{v\} \in C$*, at some point in time* t.
- *And* ψ *be a bracketed conjunction of truth assignments that* $\mathcal{A}$ *has already selected for variables in* C *at time* t*, such that:*

- ***An outer conjunction**, θ, of (1)ρ, **AND** (2)ψ, forces C to evaluate to* False.
- *Then, ρ is an* Unassignable Statement.

Definition 3.12. Constraint Function*: A (negatively defined) Constraint Function $\mathcal{F}$, at some point in time t, for a clause C, maps to the conjunction γ, of truth assignments, that is disallowed at time t, for a satisfying algorithm $\mathcal{A}$ to C.*

The above $\implies$ $\mathcal{F}$ records *γ, which is an **an outer conjunction** of (1) The conjunction of negations $\neg\psi$ of assignments ψ that $\mathcal{A}$ has already made at time t **AND** (2) ρ, the unassignable statement of C at time t.*

We elucidate this further, along with the notion of a "recording", for both the constraint function (as a function) and the unassignable statement (as a statement), with the following examples and remarks:

Example 3.6. *In the clause $(x_2 \vee \neg x_3 \vee \neg x_4)$, the conjunction $-2 \wedge 3 \wedge 4$ should be "recorded" as an unassignable statement for the clause at time $t = 0$ because the statement $-2 \wedge 3 \wedge 4$ does not satisfy the clause at any time. At time $t = 0$, this statement is also the constraint function. Thus, at time $t = 0$, the unassignable statement coincides with the constraint function.*

Example 3.7. *In the clause $(x_2 \vee \neg x_3 \vee \neg x_4)$, if at time $t = 1$, an algorithm has already selected -2, then the conjunction $3 \wedge 4$ **in conjunction with** -2 makes the clause unsatisfiable and so $3 \wedge 4$ should be recorded as an unassignable statement at time $t = 1$. Note that the constraint function itself now maps to/records the statement $2 \wedge 3 \wedge 4$ at time $t = 1$, as defined.*

Example 3.8. *In the clause $(x_2 \vee \neg x_3 \vee \neg x_4)$, if at time $t = 2$, an algorithm has already selected -2 and -3, then the clause is satisfied by the selection of -3 and so no statement is recorded for the unassignable statement at time $t = 2$. Note that the constraint function itself now maps to/records the statement $2 \wedge 3$ at time $t = 2$, as defined.*

Note Correspondence: Negatively defined constraint function/unassignable statements. In proof complexity and SAT solving, these correspond directly to the notion of forbidden partial assignments (also called conflict clauses or nogoods). We recast the (combination of the) terms in calculus terminology so that the accumulation of disallowed moves now functions as a coefficient (Section. 3.4.2) within a differential system, while retaining a terminological correspondence with the holographic literature. This allows us to parallel SAT solver propagation, with the removal of trajectories by constraints within a deterministic, dynamical system.

Remark 3.3. *A clause C* must *always have an associated constraint function. On the other hand, if an algorithm has already made a satisfying assignment to a variable in C, then the unassignable statement is empty, by definition. Thus, unassignable statements are always associated one-to-one with a constraint function, and we say that the unassignable statement is recorded to the constraint function.*

Remark 3.4. *Just as we did for a DetAlg, we will represent an unassignable statement by a singly quoted, comma delimited string, using Convention. 3.1. For example, the unassignable statement, U, recorded to the constraint function $\mathcal{F}$ shown in the above Example. 3.6 at time $t = 0$, is represented as '$-2, 3, 4$'. We* **define** *this representation as the* representation *of U and the* explicit representation *of $\mathcal{F}$. $\mathcal{F}$ can also have an implicit representation. Hence, the representation of an unassignable statement is the* **explicit representation** *of the associated constraint function $\mathcal{F}$. $\mathcal{F}$ is* **implicitly represented**, *as defined, by the set of assignments ψ that have been selected.*

Keep in mind that all the propositions proved in this section are scoped to a 3-SAT formula with exactly one clause, that is, a *small system*. For the stated propositions, let $\mathcal{A}$ be a proof system, that is, an algorithm which implements the proof calculus.

Proposition 3.1. *For a 3-SAT clause C, at time $t = 0$, there is exactly one unassignable statement U. U is the statement which assigns every variable in C to a value so that C is not satisfied (see Example. 3.6). $\mathcal{A}$ must record U to a constraint function, $\mathcal{F}$, for C.*

Proof. If we allow $\mathcal{F}$ to execute U, *then* C cannot be satisfied. But we know that a small system is *always* satisfiable, therefore, $\mathcal{A}$ must record U to $\mathcal{F}$, for C. □

Proposition 3.2. *For a 3-SAT clause C, at time $0 < t < 3$, if we assign one of the truth assignments χ, recorded in the current unassignable statement, U, of constraint function $\mathcal{F}$, of C, and C is not yet satisfied by any assignment made at prior times $< t$,* then *U must be replaced by $\mathcal{A}$, with a new unassignable statement $U^{'}$, constructed as follows:*

- *Delete χ from U.*
- *Assign the rest of the statement of U (with χ removed) to $U^{'}$.*

Proof. Since χ has already been assigned, if we allow execution of the rest of the content of the current unassignable statement U (excluding χ), then C cannot be satisfied. Since C is satisfiable, $\mathcal{F}$ *must* be updated by $\mathcal{A}$, with a new unassignable statement, that is, $U^{'}$. □

Example 3.9. *In the clause $(x_2 \vee \neg x_3 \vee \neg x_4)$, the conjunction $-2 \wedge 3 \wedge 4$ is a valid unassignable statement at $t = 0$, because it does not satisfy the clause at any time. The associated constraint function is explicitly represented as '$-2, 3, 4$'. If at time $t = 1$, $\mathcal{A}$ assigns x_2 to* false *(-2 is selected), then $\mathcal{A}$ must update the record (unassignable statement) of the constraint function to '$3, 4$'.*

Proposition 3.3. *For a 3-SAT clause C, at time $t = 3$, if C is not yet satisfied by any assignment made at prior times < 3 and there is only one variable γ left to be assigned,* then, *γ must be assigned by $\mathcal{A}$, to the truth value which satisfies C, that is, forces C to evaluate to* True.

Proof. If $\mathcal{A}$ does not assign γ to a satisfying truth value, C cannot be satisfied, and since we know that C is satisfiable, $\mathcal{A}$ *must* assign γ to the truth value, which forces C to be satisfied. □

Example 3.10. *In the clause $(x_2 \vee \neg x_3 \vee \neg x_4)$, The conjunction $-2 \wedge 3 \wedge 4$ is a valid constraint function at $t = 0$. because it does not satisfy the clause at any time. The associated constraint function is explicitly represented as '$-2, 3, 4$'. If at time $t = 1$, $\mathcal{A}$ assigns x_2 to false (-2 is selected), and at time $t = 2$, $\mathcal{A}$ assigns x_4 to* true *(4 is selected), then $\mathcal{A}$ must immediately assign x_3 to* false *(select -3) so that the clause is satisfied.*

The rules below are *actional/operational* variants of the corresponding propositions above, that ***must*** be implemented by *any* proof system of our calculus.

Note Correspondence: The rules can be read as instances of the unit propagation rule used in automated theorem proving [21, 22, 23]. We use them in the same vein, but with slightly different semantics — so that the assignment enforced by each rule is deterministic for *differentiating proofs*, in addition to satisfying the underlying clause — that is, assignments function within the context of *differentiating* a proof.

Rule 3.1. *For a 3-SAT clause C, at time $t = 0$, there is exactly one unassignable statement U, U being the statement which assigns every variable in C to a truth value ρ, so that C remains unsatisfied (see Example. 3.6).* **Record** *U to a constraint function $\mathcal{F}$, for C.*

Rule 3.2. *For a 3-SAT clause C, at time $0 < t < 3$, if we assign one of the values χ, recorded in the unassignable statement U for the constraint function $\mathcal{F}$ of C, and C is not yet satisfied by any assignment made at prior times $< t$, then* **replace** *U with a new unassignable statement $U^{'}$, constructed as follows:*

- *Delete χ from U.*

- *Assign the rest of the statement of U (with χ removed) to $U^{'}$.*

Rule 3.3. *For a 3-SAT clause C, at time $t = 3$, if C is is not yet satisfied by any assignment made at prior times < 3 and there is only one variable γ left to be assigned,* **assign** *γ to the truth value which forces C to be satisfied.*

We can now, with this, define the differentiation for a *single clause*, algorithmically. **Note** that this is a ***working definition*** which will be updated in Section. 3.4.9, using an algebraic, rather than purely algorithmic formalism.

Definition 3.13. Partial Differentiation*: For a clause C, at any point in time, $t_i \geq 0$ where i number of variables have been assigned to truth values, $i < 3$ — any procedure that has enforced Rules. 3.1 and 3.2 at all points in time t_x, in the range $0 \leq x \leq i$, is a* Partial Differentiation *of the solution space, θ, of C.*

Definition 3.14. Partial Derivative*: A solution computed by a* Partial Differentiation *is a* Partial Derivative*.*

Definition 3.15. (Total) Differentiation*: For a clause C, at the point in time (t_4) when we have a complete solution to C, that is, when all 3 variables in C have been assigned truth values — any procedure that has enforced Rules. 3.1 and 3.2 at all points in time in the range $0 \leq t < 3$ and Rule. 3.3 at time $t = 3$, is a* (Total) Differentiation *of the solution space, θ, of C.*

Definition 3.16. (Total) Derivative*: A solution computed by a* (Total) Differentiation *is a* (Total) Derivative*.*

3.4.2 Formal Notion Of A Derivative

In order to complete the formal definition of differentiation for a single clause, we take a step ahead and define what it means to *functionally* compute a derivative for a formula F_n, of arbitrarily many clauses and variables.

We define an analog of the solution space for F_n. Note that unlike the case for a single clause, we do not supply an exact number for the elements of the solution space of F_n, but only note that it must have some cardinality:

Definition 3.17. Solution Space Of A 3-SAT Formula, θ_{F_n} *: Let $V = v_1, \ldots, v_n$ be the set of variables in a 3-SAT formula F_n with n variables.*
Then θ_{F_n} for F_n is the structure given by:

- *The set of all satisfying solutions (sets of truth value assignments) to F_n, S_n : $\{v_1 := 0 \oplus 1, \ldots, v_n := 0 \oplus 1\}$.*
- *And equipped with the operation of formula level assignment, $[:=]^n : [0,1]^{|S_n|} \to [0,1]$.*

Definition 3.18. Internal Nodes Of θ_{F_n} *: Let F_n be a 3-SAT formula, and:*

- *S_n be the set of all satisfying solutions $\{A_k\}$ to F_n, A_k : $\{v_1 := 0 \oplus 1, ..., v_n := 0 \oplus 1\}$.*
- *And $\int_i \subseteq S_n$.*

Then, $\int_i$ is an internal node *of the solution space θ_{F_n}, of F_n.*

Remark 3.5. *A DetAlg $\mathcal{D}$ is an element of the solution space of F_n: $\mathcal{D} \in \theta_{F_n}$ (and of θ for a formula with one clause) since $\mathcal{D}$ computes an internal node.*

Remark 3.6. *For the definition below and going forward, assume that the constraint function for F_n is just a collection (set) of the constraint functions of its individual clauses and its representation is as well just a collection of the representations of the individual constraint functions. This assumption will be demonstrated to be correct when we define the operators for a large system.*

Let $\mathcal{A}$ be an algorithm which creates a valid unassignable statement for all clauses in F_n at time $t = 0$ and differentiates all clauses in F_n according to Definitions. 3.13 and 3.15, at all points in time, $t_i, 1 \leq i \leq n$, where n is the number of variables in F_n.
And $g(t_i)$ be the constraint function of F_n, parameterized by time, at time t_i.
And f be the function (allowed motion) computed by $\mathcal{A}$, that is, a *DetAlg*.
And θ_{F_n} be the solution space of F_n as defined at the beginning of the section.
Then the following definitions hold:

Remark 3.7. *In these definitions, we refer to all differences between successive quantities of some measurement as* differentials *because they are obtained as parts, numerators/denominators, of a rate of change determination — Definitions. 3.24 and 3.25.*

Definition 3.19. Block Of Time*: A* Block Of Time *B is defined as either:*

- *A single, free assignment a, which is not immediately followed by any bound assignment,* ***OR****.*
- *A set of assignments,* $A = a_i, \ldots, a_j \quad 1 \leq i < n, i < j \leq n$*, selected in increasing order, where* $|A| > 1$*,* a_i *is a free assignment and* $a_{i+I}, \ldots, a_j$ *are bound assignments,* $i + 1 \leq j$*.*

Remark 3.8. *The next available point in time, for a free assignment, after B is executed, is* $t = i + |A|$ *which is also the same as* $t = j + 1$ *(since points in time* $\in \mathbb{N}$*).*

Any assignment made for a single clause as a result of the enforcement of Rule. 3.3 at time $t = 3$, forms a block of time with the assignment made at time $t = 2$. For a more complex example of a formula with 3 clauses, see below:

Example 3.11. *Let F be a 3-SAT formula which is a conjunction of* 3 *clauses and which at time* $t = 0$*, has the following set of unassignable statements: '*$1, -2, 3$*', '*$1, -2, 4$*' and '*$-3, -4, -5$*', so that the selection of assignment* 1 *at time* $t = 1$ *and assignment* -2 *at time* $t = 2$ *force the selection of assignments* -3 *and* -4 *which together force the selection of assignment* 5*. We say that the set* $\{-2, -3, -4, 5\}$ *forms a block of time b at time* $t = 2$*. If F had more clauses, the next available point in time would be at* $2 + |\{-2, -3, -4, 5\}|$ *which is at time* $t = 6$*.*

Note Correspondence: The notion of a block of time is structurally analogous to implication chains in unit propagation: a single free assignment deterministically triggers several bound ones. We frame this temporal grouping as an interval so that first- and second-order differentials can be defined, allowing the proof calculus to perform *classical* analysis so as to determine *all* ***logical*** assignment implications in a chain, for a 3-SAT assignment procedure.

Definition 3.20. First-Ordered Partial Time Differential $\Delta t_{(1)}$*: A* First-Ordered Partial Time Differential $\Delta t_{(1)}$ *is the duration measured by the calculus between* 2 *successive points in time,* t_i *and* t_{i+1}*, where* $1 \leq i \leq n$*, that is,* 0 *or more assignments are remaining to be made after* t_i*. The last assignment made at time* t_n *is also a* First-Ordered Partial Time Differential*.*

Note that t_{n+1} denotes the time *after* truth assignments have been selected for all n variables.

Definition 3.21. Second-Ordered Partial Time Differential $\Delta t_{(2)}$*: A* Second-Ordered Partial Time Differential $\Delta t_{(2)}$ *is the duration measured by the calculus between* 2 *blocks of time,* b_1 *and* b_2*, where* b_2 *may contain* 0 *or more assignments, that is,* 0 *or more assignments are remaining to be made after the completion of* b_1*. The last assignment made at time* t_n *is also a* Second-Ordered Partial Time Differential*.*

Remark 3.9. *A time differential is partial because the* 2 *endpoints,* 2 *points in time, act as independent variables (can be given in any order, at least for a formula with just* 1 *clause) in a solution derivation by* $\mathcal{A}$*.*

Note Correspondence: First- and second-order time differentials introduce a concept of duration not used in traditional SAT solving and could be regarded as corresponding to propagation depth and derivation depth as used in the proof complexity of such solvers. We construe them here as infinitesimal time intervals in order to define deterministic rates of change over proof trajectories.

Definition 3.22. Measure Function Differential*:*

1. *Let* Γ *be the set of all complete sets of assignments (corresponding to internal nodes* N *of* θ_{F_n} *where* $|N| = n$*), that are (still) assignable (*$\mathcal{A}$ *can approach* N*) at some time* t*. That is, elements of* Γ *are solutions with* n *complete assignments that can be assigned to* F_n *based on the* **total representation** *of the constraint function of* F_n *at time* t*.*
2. *Define* $\mu(t)$ *as the size of* Γ*:* $|\Gamma|$*.*
3. F *is a* counting measure *differential,* $d(t,\ t+1)$*, defined as the difference (differential)* $\Delta\mu$*, in* $\mu(t)$*, measured by the calculus, between* 2 *consecutive points in time,* t *and* $t+1$*.*

We ***define*** F *as the* Measure Function Differential*.*

Note that we define F between points in time and not between blocks of time.

Definition 3.23. Coefficient of Differentiation*: Each* $g(t_i)$ *is a coefficient of the derivation of the* i^{th} *assignment — it corresponds with the precomputed anti-derivative* $a_i \equiv d_{i-1}$*, of* d_i*, where* d_i *is a denotation of the* i^{th} *(partial) derivation. Thus, each* $g(t_i)$ *is the* coefficient *of differentiation of* θ_{F_n} *at time* t_i*. We* ***define*** $g(t_i)$ *as the* i^{th} Coefficient of Differentiation*.*

Remark 3.10. *The above definition can be interpreted as a requirement that the constraint function (for an arbitrary number of causes) must always correspond to the antiderivative of a differentiation in the calculus.*

Note Correspondence: Constraint functions as coefficients of differentiation parallel the way invariants constrain transitions in SAT solving as the constraint is always present.

Definition 3.24. First-Order Instantaneous Rate Of Change Of The Measure Function*:*

1. *At each point in time* $t_i, 1 \leq i \leq n$*, the size of* $\mu(t_i)$ *(Definition. 3.22) decreases exponentially from the value it had at* $\mu(t_{i-1})$*, so that the following holds:*

$$\lim_{\mu(t)\to 1} \frac{\mu(0)-1}{\Delta t_{(1)}} = 0.$$

2. $\Delta t_{(1)}$ *is a non-subdividable instant of time, with no observable/measurable internal first-order duration.*[3]
3. *Recall that* F *is defined as* $\Delta\mu$ *and so* ΔF *is the change in* F *at each point in time w.r.t.*

We ***define***

$$\frac{\Delta F}{\Delta t_{(1)}}$$

as the First-Ordered Instantaneous Rate Of Change Of The Measure Function*.*

Definition 3.25. Second-Order Instantaneous Rate Of Change Of The Measure Function*:*

1. *At each block of time* b_k*, the size of the set of distinct points in time* $\{t_i\}$ *included in* b_k *is determined by* $\mathcal{A}$*, and the value of the measure function* $\mu(t)$ *at each* t_i *decreases exponentially from the value it had at time* t_{i-1}*. This determines the rate of change* R^2*, of the following limit:*

$$\lim_{\mu(t)\to 1} \frac{\mu(0)-1}{\Delta t_{(1)}} = 0.$$

[3]We prove in A.1 that $\Delta t_{(1)} \in \mathbb{C}$.

2. *So that* R^2 *takes the form:*

$$\lim_{\mu(t)\to 1} \frac{\left(\lim_{\mu(t)\to 1} \frac{\mu(0)-1}{\Delta t_{(1)}}\right)}{\Delta t_{(2)}} = 0.$$

3. $\Delta t_{(2)}$ *is an instant, non-subdividable block of time, with no observable/measurable internal second-order duration.*

R^2 *measures the rate of change of the* First-Order Instantaneous Rate Of Change Of The Differential Measure *with respect to the second-ordered time differential. We* ***define***

$$\frac{\Delta^2 F}{\Delta t_{(1)} \cdot \Delta t_{(2)}} \equiv \frac{\Delta^2 F}{\Delta t^2}$$

as the Second-Ordered Instantaneous Rate Of Change Of The Measure Function*.*

Theorem 3.2. *Let* $\mathcal{A}$ *be defined as at the beginning of this section and* $\mathcal{A}(t_i)$ *be a step in* $\mathcal{A}$ *executed at time* t_i*. Then:*

$$\mathcal{A}(t_i) \equiv \mathcal{F}\left(g(t_i), \frac{\partial^2 F}{\partial t_i^2}\right). \tag{3}$$

Proof. $\mathcal{A}(t_i)$ computes a second-order differentiation

$$\mathcal{F} : \frac{\partial^2 f}{\partial t^2} = \lim_{\delta(t)\to 0} \frac{\mu(t_{(1)}+1) - \mu(t_{(1)})}{\delta(t)}.$$

where $\delta(t)$ is $\Delta t_{(1)} \cdot \Delta t_{(2)} \equiv \Delta t^2$. The coefficient of this differentiation is $g(t_i)$. Therefore, Eqn. 3 holds. □

Note: $\mathcal{F}$ is a differential operator, that is, a function of differentiation.

Remark 3.11. *The function being differentiated is a DetAlg — the differentiation computes the second-order rate of change of truth assignments that compose the DetAlg.*

From this remark, it is appropriate to say, as a shorthand expression, that we differentiate the solution space θ_F of a formula/clause, since a *DetAlg* $\in \theta_F$ and therefore represents θ_F along any trajectory of differentiation.

We now decompose each algorithmic step $\mathcal{A}(t_i)$ into set-theoretic functions, for use in further algebraic analysis of the calculus:

Let *DF*, defined over $\mathbb{R}$, denote the set of derivable functions, and *CF*, defined over $\mathbb{C}$, denote the set of underivable functions, for some formula F at some time t. By derivable, we mean a function that can be computed as a (total or partial) derivative.[4] *CF* corresponds to the constraint function/set of all constraint functions of all (an arbitrary number of) clauses contained in F.

Definition 3.26. $\delta(t)$-function*: Computing the differentiation function at a point in time, is a function which maps CF to CF, that is, it operates over the domain of the coefficients of differentiation.* $\delta(t)$ *is thus, a differential operator, that is, a function of differentiation.*

$$\delta(t) : \mathbb{C} \longrightarrow \mathbb{C}, \quad \delta(t) = \mathcal{F}(g(t), \frac{\partial^2}{\partial t^2}).$$

Note Correspondence: Pruning or updating of constraints in SAT solving.

[4]We prove the number-theoretic structure of these domains in A.1 and A.2, after setting up the necessary theoretical framework needed to prove them, in the rest of the paper

Definition 3.27. λ-function*: The function f_t, which records a truth value to a variable at some point in time t, is a derivative at t. The function λ, which computes f_t, maps DF to DF. λ is the differentiation function (of f_t) and corresponds with the positive trajectory of truth assignment.*

$$\lambda : \mathbb{R} \xrightarrow{\delta(t)} \mathbb{R}, \quad \lambda(f_t) = f' = \frac{\partial^2 f_t}{\partial t^2} = f_{t+1} \equiv \textit{DetAlg,} \text{ at time } t+1.$$

Every f_t is also an antiderivative of all its (possible) successors at time $t+1$. f_t as an antiderivative, at all times $t > 0$, is computed by the λ-function of its predecessor at time $t-1$. At time $t = 0$, this antiderivative is total, and partial at all other times $t > 0$. As such, every λ function is also an integration:

$$\lambda : \mathbb{R} \xrightarrow{\delta(t)} \mathbb{R}, \quad \lambda(f_{t-1}) = \int f_{t+1}\, dt = f_t \equiv \textit{DetAlg,} \text{ at time } t.$$

Note Correspondence: As differentiation, corresponds to propagation of assignments (derivation) in SAT solving.

Remark 3.12. *$\delta(t)$ is also as a consequence, a function of integration: $\mathcal{F}(\int \frac{\partial^2}{\partial t^2}\, dt)$ (in addition to differentiation, readily seen in the case of a formula with one clause). It establishes CF at time $t = 0$.*
***Note Correspondence**: Recording conflicts (anti-derivation) in SAT solving.*

Note: f_t in the case of integration, is a "volume" element over all of its possible *directional* derivatives (positive trajectories) at time $t+1$, of which one will be selected — *along* with (sheaf) data about impossible derivatives.[5] Therefore, at any time $t, t < n$, for a formula F_n with more than one solution, 2 or more instances of f_{t+1} are mapped (by $\lambda(f_{t-1})$ for $t > 0$) to f_t.

For the majority of the time, in this article, when speaking of λ-functions, we will focus on their behavior as a differentiation operation — because of its simpler, more straightforward description — and always refer to integration as dual to differentiation as a λ-function.

Definition 3.28. σ-function*: Marking a node (set of assignment values) as unreachable, at some point in time, t, is a complement of differentiation, namely, a codifferentiation. The function σ, which computes this codifferentiation maps DF to CF and corresponds with the negative trajectory of disallowed truth assignments. As a vector form, it is a covector to the λ-function.*[6]

$$\sigma : \mathbb{R} \xrightarrow{\delta(t)} \mathbb{C}, \quad \sigma(f_t) = \bar{f}_t.$$

This function will be useful whenever we describe (multi)linear forms/products on the space of the calculus.

Remark 3.13. *The motion of $\delta(t)$ itself can be associated with a (unrepresented) dual-valued metric ϵ on the space of motions of the calculus, since its effective displacement at each point in time is always 0 with respect to the displacements of the other 2 functions, and since each time, the number of derivable functions is halved, so that $\epsilon^2 = 0 \implies \epsilon$ is a dual number.*

Our treatment of a meta-algorithm $\mathcal{A}$ which differentiates another *continuously-valued* algorithm β, has precedence in other works that discuss the derivatives of algorithms, notably in the context of machine learning [24, 25, 26, 27]. In this sense, $\mathcal{A}$ can be interpreted as automatically differentiating β by storing the (unrepresented) value of a (hyper-) dual metric [28] which functions as a (second, outer) coefficient of the *function of composition* of the $\delta(t)$ operators.

[5] f_t is a *blowup* at t.

[6] For the interested reader: As such, algebraically, whenever we identify an algebra on the space, it also corresponds with a coalgebra, and thus, the calculus as an algebra, is always a bialgebra, in which the differentiation fragment is a unital associative algebra over a ring and the codifferential fragment is its dual counital coassociative coalgebra. Thus, technically, any associated algorithm is a bialgorithm and the $\delta(t)$ is a bioperator.

3.4.3 The Underlying Logical Structure Of A Derivative

Recall that we expressed the Holant Reduction as a predicate of *motion* in Expr. 2. Here we do the same for a formula with n variables in our calculus.

First, we note that solutions to a 3-SAT formula can be regarded as belonging to the order of propositional logic, that is, the level of the 3-SAT formula. They are simply conjunctions of assignments, which we have called, the *content* of a *DetAlg*. We refer to this as the 0^{th}-order. Thus, a *DetAlg* is a 0^{th}-order object.

Next, we note that λ functions, which compute a *DetAlg*, are a model for the *DetAlg*. This model is restricted by an implicit predicate which can be expressed in the form of Expr. 2, and so it is a first-order model (not to be confused with the fact that it is a second-order differentiation). λ functions are themselves ordered externally, by $\delta(t)$, according to the following formula:

Let θ_{F_n} be the solution space of some of some 3-SAT formula F_n.
Let $\mu(\theta_{F_n})$ be the number of solutions to F_n.
Let $\{V_1^k....V_n^k\}, 1 \leq i \leq n$, be time-indexed internal nodes of θ_{F_n} along some trajectory $k, 1 \leq k \leq \mu(\theta_{F_n})$.
Let g_{t_i} be the coefficient of differentiation at time t_i along this trajectory.
And let $f(V_i^k)$ be the *DetAlg* that computes the value of V_i^k at time t_i. (Remark. 3.11).
Then:

$$(\forall k \wedge (\forall V_i^k \subset \{V_1^k....V_n^k\}), \quad M(V_i^k, V_{i+1}^k) := \begin{cases} \top & \exists \delta(t_i), \quad \delta(t_i) = \left(g_{t_i}, \dfrac{\partial^2 f(V_i^k)}{\partial t_i^2}\right) := V_{i+1}^k. \\ \bot & \text{otherwise.} \end{cases} \tag{4}$$

As a logic then, the above formula is a model for λ functions, that is, a meta-model for a *DetAlg*. It is a second-order structure. M here, is a second-order predicate of motion computed by a meta-algorithm $\mathcal{A}$ and then used to functionally order λ functions (differentiation) via an integro-differential operator $\delta(t)$. This places a $\delta(t)$ function, by the logic of its chain of precompositions and compositions, as a second-order *logical* transition (function) between differentiations.

3.4.4 Basic Geometric And Topological Notions

We introduce some basic geometric and topological notions on our space.

Proposition 3.4. *A λ function operates in a metric space $M(\lambda)$.*

Proof. . We defined the operation of a λ function as a motion of an assignment trajectory in some differentiable solution space, θ_{F_n}. We define the Euclidean space of the motion of λ as $M(\lambda)$.
We can define a distance function on $M(\lambda)$, $d : d(\lambda(t_1), \lambda(t_2)) = 1$ between any 2 distinct points in time.
This turns $M(\lambda)$ into a metric space. □

Proposition 3.5. *$M(\lambda)$ has a measure.*

Proof. At any point in time t, $M(\lambda)$ has μ number of possible trajectories, that is, λ functions. Let $P(\lambda_i, t)$ be the probability that some trajectory λ_i is selected at time $t+1$, then we can associate a probability measure $\mu(p, t)$ to $M(\lambda)$ at some time t, which equals the total sum of these probabilities : $\mu(p, t) = \sum_i P(\lambda_i, t)$.
This defines a measure on $M(\lambda)$. □

Proposition 3.6. *$M(\lambda)$ is a measure-preserving space.*

Proof. . The measure $\mu(p, t)$ on $M(\lambda)$ is always 1 at any point in time t. □

Proposition 3.7. *$M(\lambda)$ is a normed space.*

Proof. We can define a norm on $M(\lambda) : |\lambda(t)| = t$ since the metric (induced by truth value assignment) adds unit length. □

Theorem 3.3. *$S(\lambda(t))$, the set of all time-indexed points traversed by a λ function, that is, points on the trajectory of the λ function, is an open set.*

Proof. Let $S(\lambda(t_i)) = \lambda(t_i) \mid t_i \in T$. For each $\lambda(t_i), \exists \epsilon > 0$ s.t $\forall \lambda(t_{i+1}) \in M(\lambda)$, $(d(\lambda(t_i), \lambda(t_{i+1})) < \epsilon \Rightarrow \lambda(t_{i+1}) \in S(\lambda(t))$. Since sequences of $\lambda(t)$ are discrete and ordered, each $\lambda(t)$ has a neighborhood bounded by such an ϵ. Thus, $S(\lambda(t))$, by definition, is open. □

Theorem 3.4. *Let SF be the set of all possible trajectories of λ at some point in time , then SF is a discrete topology.*

Proof. *SF* has the following properties:

- Each singleton $S(\lambda(t)) \subseteq SF$ is open.
- The union of all $S(\lambda(t))$ is *SF* and is included in *SF*.
- The finite intersection of all $S(\lambda(t))$ is the null/empty set $\emptyset$ and is included in *SF*.
- All subsets of *SF* including $\emptyset$ and *SF* are open.

Hence, *SF* satisfies the definition of a discrete topology. □

Remark 3.14. *The space of the operation of λ functions, that is, differentiation, is a complex differentiable manifold M_T. Simply note that M_T charts the progression of real-valued functions — λ and σ functions — with operators that are arguments of complex-valued functions (of time) — $\delta(t)$ functions.*[7]

3.4.5 Topological Arithmetic

Here we define the notion of arithmetic operators defined over a topological space. These operators will help us properly define the algebraic structures needed for the calculus.

We define the following arithmetic operations : multiplication and addition, for a topological space, along with their respective (algebraic) structure-related properties. We start with multiplication:

Definition 3.29. Multiplication $\times$ *: A product on a topological space is defined as the coarsest topology on the space, that is, the topology which induces the fewest open sets. Multiplication of 2 open sets S_i and S_j — $S_i \times S_j$ — in a topology T is thus **defined** as the removal/elimination of both S_i and S_j from T.*

Definition 3.30. Multiplicative Identity *: A single open set S is trivially identical to itself. Removing S from a topology T that contains just one subset S, excluding the entire set and the empty set, is equivalent to removing both S and the boundary, which we call $\mathcal{Z}$, from T. The boundary is always removed from T by definition and so the following is true: $S \times \mathcal{Z} = S$. Thus, we **define** $\mathcal{Z}$ as the multiplicative identity of topological multiplication.*

A structure G defined with such a topological multiplication has the following algebraic properties for multiplication:

- Closure : $\forall S_i, S_j \in G, S_i \times S_j \in G$
- Associativity : $(S_i \times S_j) \times S_k = S_i \times (S_j \times S_k)$.
- Identity : $\exists S_1 \in G$ s.t. $S_i \times S_1 = S_i$. S_1 being $\mathcal{Z}$.

[7] M_T can be regarded as a transverse manifold over (cobordism of) ambient manifolds $M_X(t)$ at successive points t in time.

- Inverse : $\exists S_i^{S_{-1}} \in G$ s.t. $S_i \times S_i^{S_{-1}} = S_1$.
  The inverse operation is not (well) defined for (open) sets in a topology and we have no particular intuition for its implementation, but the following holds:
  That S_{-1} is the *additive* inverse of S_1 is trivial to establish.
  We rewrite the statement as $S_i \times \frac{1}{S_i^{S_1}} = S_1$.
  Multiply each side by $S_i^{S_1}$: $S_i = S_1 \times S_i^{S_1}$.
  Since S_i is a real number (argument of a real-valued function — Appendix Theorem A.2), its multiplicative identity S_1 is 1.
  And so, we get (simplifying just the exponent), $S_i = S_1 \times S_i^1$, which gives us back the identity, and so is a tautology.

We restrict the addition operation to addition defined over a *discrete topology*, which is sufficient for our purposes.

Definition 3.31. Discrete Topological Addition $+$ *: The topological sum of* 2 *sets* S_i *and* S_j *is defined as their disjoint union,* $S_i \sqcup S_j$*, in a topology* T*, which for a discrete topology is their direct sum,* $S_i \oplus S_j$*, as singletons. Addition of* 2 *sets* S_i *and* S_j *—* $S_i + S_j$ *— in a discrete topology* T *is thus* ***defined*** *as the inclusion of the disjoint union which is also the direct sum, of both* S_i *and* S_j *in* T*.*
Note *that topological multiplication for a discrete topology as we defined it, implies topological addition — to remove an arbitrary open set from a discrete topology t, we must remove it from a set* S*, the union of all open sets, which is also the disjoint union of all subsets of the topology of* T*, excluding* $\emptyset$ *and the entire set (union of all sets).*

Definition 3.32. Discrete Topological Additive Identity *: A single open set* S *is trivially identical to itself. Constructing a discrete topology* T *that has just one subset* S*, excluding the entire set and the empty set* $\emptyset$*, is equivalent to including the disjoint union of both* S *and the empty set, which is just* S*. This means* $S + \emptyset = S$*, Therefore, we* ***define*** $\emptyset$ *as the additive identity of discrete topological addition.*

A structure G defined with such an addition, is always defined ***along a path*** in a topological space equipped with T, and has the following algebraic properties:

- Closure : $\forall S_i, S_j \in G, S_i + S_j \in G$
- Associativity : $(S_i + S_j) + S_k = S_i + (S_j + S_k)$.
- Identity : $\exists S_0 \in G$ s.t. $S_i + S_0 = S_i$, S_0 being $\emptyset$.
- Inverse : $\exists (S_i \times S_{-1}) \in G$ s.t.
  $S_i + (S_i \times S_{-1}) = S_0$, as follows:
  Divide all terms in the above equation by S_{-1} to get $\frac{S_i}{S_{-1}} + S_i = \frac{S_0}{S_{-1}}$, which gives $(S_i \times S_1) + S_i = S_0 \times S_1$.
  The R.H.S of this equation removes $\emptyset$ from a disjoint union with S_i, which is equivalent to removing S_i from the topology since $\emptyset$ must always be included in the topology.
  Which can be rewritten, because of the definition of multiplication, as $\emptyset + S_i = S_i$, which is the same as $S_0 + S_i = S_i$, which gives us back the identity, and so is a tautology.

3.4.6 Twisted Operators And Structures

Here we define the notion of twisted arithmetic operators and structures. This concept is also useful for defining the algebraic structures used by the calculus.

Definition 3.33. Twist *: Let the* <u>usual</u> *arity of an operation* A *be* k *and* A^k *represent the definition of the operation, then a mapping*

$$\oplus_n : A^{k+n} \to A^k$$

twists A *if* $n > 0$*. An untwist sequence* τ *reduces* n *to* 0 *and restores* A *to its normal arity.*

For example, an addition operation is twisted if it is defined for 3 or more arguments. An untwist sequence must then remove extra arguments (in some specifiable order), until only 2 arguments are left, after which a result is obtained as per the usual definition of addition.
A *twisted structure* is one in which one or more of its defining operations is twisted.
The following twisted operators are twisted topological operators. .

Definition 3.34. Twisted Multiplication T(×)*: Removal of more than* 2 *open sets from a topological space t is* Twisted Multiplication T(×). Twisted Multiplication T(×) *has the same algebraic properties as non-twisted multiplication, via induction from a base case that uses non-twisted multiplication.*

Definition 3.35. Twisted Addition T(+)*: A disjoint union/direct sum of more than* 2 *open sets in a topological space t is* Twisted Addition T(+). Twisted Addition T(+) *has the same algebraic properties as non-twisted addition, via induction from a base case that uses non-twisted addition.*

Remark 3.15. *Twisted operations can be interpreted as quantum operations, based on the following criteria, stated informally:*

- ***Superposition*** *: They can be expressed as a superposition.*
- ***Entanglement*** *: Interactions of operations over different clauses are "entangled" (that is, if they can be composed into an ultraproduct, which we show is possible).*
- ***Chirality*** *: We show in Section. 3.4.9 that these operations are over so-called Chiral Objects corresponding to a Chiral Algebra [29], shown by the authors of the cited work, Beilinson and Drinfeld, to be quantum objects.*

Structures with twisted operations as we defined the term, can be interpreted as quantum algebraic structures. We defer the formal explication of these operators in terms of their full quantum-theoretic formulation, to future work, such an effort being out of scope for the current one.[8]

3.4.7 Algebra Of Differentiation — Rings

We now discuss the ring structures associated with the calculus. The definitions and theorems in this section relate to a formula F_n with n variables and solution space θ_{F_n}. They constitute a mid-step towards generalizing our procedure to a larger, predicative system of clauses.

As noted earlier (Remark. 3.6), we denote the collection of all clause-level constraint functions of clauses in F_n, as the constraint function of F_n. We also introduce algorithmic reinterpretations of $\delta(t)$-functions as well as an alternative description of a *DetAlg*. The introduced terms fit the definitions of the functions into a holographic framework, so that we can use these terms alternatively, in the context of the ***flow*** and gadgetry, of *derivations* computed by a proof system which implements the calculus. In this sense, these terms *correspond* with their purely functional counterparts.

Definition 3.36. Generator*: Every time-indexed constraint function CF_i of F_n is the* Generator *(source) of a (second-order partial) differential $\partial^2 f$ of θ_{F_n} at some point in time t_i, as a coefficient of differentiation. We denote this differential as ∂_i.*

[8] From this informally stated equivalence, we can still infer a pointwise direct logical correspondence of the algebraization of our underlying logic, with linear logic as expressed by Girard in a series of articles, starting with [30], and particularly emphasized in [31], where proofs in proof nets (whose correctness is formally interpreted by linear logic) correspond to quantum (super) operators (in a von Neumann algebra) on some Hilbert Space. Using the resource semantics reading of linear logic, we obtain the correspondence by equating the least fixed-point (which we will expand on further), at each point in time, with a resource that must be expended in a particular *direction* of differentiation. This allows us to associate the differential context around each point with a type of differentiable linear logic *DLL* and furthermore with differentiable morphisms, in the categorical extension of *DLL* (See Section. 3.4.9 for discussion of categorical semantics).

Definition 3.37. Transition Gadget*: Every (partial) differentiation* $\frac{\partial^2 f}{\partial {t_i}^2}$ *is implemented by a* Transition Gadget *acting as a conductor of a differential* ∂_i*. A* Transition Gadget *corresponds to a* δ *function and as such, also computes the antiderivative* a_i *corresponding to (where applicable) the next possibly twisted differential* ∂_{i+1}*.*

A transition gadget can have one or more "setup" procedures in order to ensure that it fulfils its role as a dual operator. These procedures will be *defined* as we define how a transition gadget relates to the algebraic definitions of differentiation and integration. These procedures will describe a transition gadget as corresponding to certain structures and states that maintain invariant properties for the differentiation and integration functions.

Definition 3.38. Recognizer*: Every time-indexed constraint function* CF_i *of* F_n *at time* $t = i$ *is the* Recognizer *(sink) of the differential* ∂_i *conducted by a Transition Gadget at time* $t = i$*. A recognizer at time* $t = n$ *is a self-recognizer.*

Note that there is (necessarily) an information-symmetry between generators and recognizers, in which every recognizer is also a generator for the next step, when applicable.

Remark 3.16. Generators *and* Recognizers *are* linear forms *introduced and eliminated by* σ*-functions. This defines the calculus, at each point in time, as a type of variational calculus.*

Definition 3.39. Constraint Quotient*: Let* G *be generators and* R *recognizers. Then:* $g_1 \sim g_2 \iff \exists r \in R$ *such that* $r(g_1) = r(g_2)$ *The quotient set* $Q = G/\sim$ *defines equivalence classes of constraints.*

Remark 3.17. *Every Transition Gadget computes a Constraint Quotient.* Constraint Quotients *are transitive by implication of the above definition, which means that, if* B *is a* Constraint Quotient *of* A *and* C *is a* Constraint Quotient *of* B*, then* C *is a* Constraint Quotient *of* A*.*

We extend the notion of derivable functions DF used in Section. 3.4.2:

Definition 3.40. Total Derivable Function — TDF *: A* TDF*, corresponding to a total derivative, can be simply denoted as a set of satisfying assignments to all variables in* F_n*, and as such, corresponds to a (completed) DetAlg.*

Definition 3.41. Time Indexed Set Of Total Derivable Functions — $\{TDF\}_i$*: A* $\{TDF\}_i$ *is the set of all* TD*s that can (still) be derived at some point in time* i *by a (deterministic) meta-algorithm/proof system that implements the calculus.*

Each $TDF \in \{TDF\}_i$, in addition to its denotation as a set of assignments, also corresponds to an open set as well as a trajectory in the topological space SF of all trajectories. Each $\{TDF\}_i$ corresponds to one generator/recognizer pair (which includes self-recognizers) and as such, a transition gadget G_i is indexed by a transition between a $\{TDF\}_i$ and its succedent $\{TDF\}_{i+1}$.

Theorem 3.5. *Each* $\{TDF\}_i$ *is defined over a Twisted Ring* $R_i(T(+, \times))$*.*

Proof. Each $\{TDF\}_i$ has both twisted addition and multiplication defined between its elements. □

Each $R_i(T(+, \times))$, where applicable, is also a subring of $R_{i-1}(T(+, \times))$, with exactly half of the elements of $R_{i-1}(T(+, \times))$ removed from $R_i(T(+, \times))$.

Next, we reformulate the set of all Transition Gadgets within the context of a vector space by the following *representational* construction:

Let A be a set of satisfying truth assignments to the variables V of some formula F. And let ζ be a sequencing of elements of A, in time:

- Each element $a \in A$, as well as unique combinations/sequences of elements $\{a_i\} \subseteq A$, expresses a motion $\mathcal{V}$ as part of the trajectory of the λ-function and can be assigned a unique metric, d, which corresponds with the metric assigned to the underlying topological space $T(\lambda)$, $\implies$ that each a, as well as $\{a_i\}$, has a magnitude.
- Each $\mathcal{V}$ partitions the set of all possible motions, $\implies$ that each a, as well as $\{a_i\}$, has a direction.
- Therefore, each a, as well as $\{a_i\}$, is a vector : $\vec{a}, \{\vec{a_i}\}$.
- ζ can also be assigned a metric (magnitude) and it also partitions the space of motions (direction), therefore ζ is also a vector: $\vec{\zeta}$.
- Let each $\vec{a}$ be assigned a unique index i, $0 \leq i \leq |A|$ and let ζ_y be a sequence whose ordering of vectors $\{a_i\}^y$ is different from the ordering $\{a_i\}^z$ by a different sequence ζ_z. But let both ζ_y and ζ_z sequence the same set of unique assignment vectors $\{\vec{a_i}\}$.
- As vectors, $\vec{\zeta_1} = \vec{\zeta_2}$ must hold since both $\vec{\zeta_y}$ and $\vec{\zeta_z}$ have the same magnitude and direction — the sequencing of assignment vectors corresponds to the addition of vectors, by definition of *vector addition* — each addition between the $\vec{a_i}$ vectors of $\vec{\zeta_y}$ and $\vec{\zeta_z}$ also corresponds to a rescaling of the associated $\vec{\zeta}$ vector and thus, doubles as *scalar multiplication* for the $\vec{\zeta}$-function.

The addition defined in the above construction, dropping the vector notation and treating vectors as integers, has the following algebraic properties:

- Closure : $\forall a_i, a_j \in \zeta, a_i + a_j \in \zeta$
- Associativity : $(a_i + a_j) + a_k = a_i + (a_j + a_k)$.
- Identity : $\exists a_0 \in \zeta$ s.t. $a_i + a_0 = a_i$. a_0 being the empty set, that is, ζ (with all elements removed).
- Inverse : $\exists (a_i \times a_{-1}) \in \zeta$ s.t. $a_i + (a_i \times a_{-1}) = a_0$, as follows - with the same argument used for topological addition, since the vector space is topological, and so we can reuse arguments featuring topological multiplication:

$$\frac{a_i}{a_{-1}} + a_i = \frac{a_0}{a_{-1}} \implies (a_i \times a_1) + a_i = a_0 \times a_1 \implies \emptyset + a_i = a_i \implies a_0 + a_i = a_i \text{ — a tautology}$$

We use this notion of addition to define a group structure on the set of all transition gadgets.

Theorem 3.6. *The set of all Transition Gadgets $\{G_i\}$ forms an additive group.*

Proof. By corresponding the set with an assignment vector space ζ with time-indexed assignment vectors $\vec{a_i}$:

- Each transition gadget corresponds to an assignment vector and sequencing of transition gadgets to form a set corresponds to the vector addition of transition gadgets, via a homomorphism:

$$\varphi : G_i \longrightarrow \vec{a_i}, \qquad \varphi(G) \subseteq \zeta.$$

Therefore, the set of transition gadgets form an additive group whose addition operation is the addition operation of ζ. □

Trivially, each subset of the set of all transition gadgets also forms an additive group. These groups are *represented* by their actions on the underlying vector space.

Remark 3.18. *The (addition) operation of this group may or may not be commutative with regards to a specific implementation of the calculus. In standard group theory language, we say the presentation of the group, via the calculus (differential time ordering), by a specific set of assignments, may or may not be commutative.*

In maximally symmetric spaces, like the solution space of a formula with a single clause, this presentation is indeed symmetric, thus commutative. In other cases, the extreme being one with a large formula and just a single solution, our calculus may decide the assignments in a strict order. In these cases, the calculus must endow the presentation of the group operation with a type of identity, which measures the exact degree, at each point in time, by which the presented group operation fails to be commutative, that is, what is called a Jacobi Identity. This Jacobi identity however, need only be implied, in the sense that the calculus is (always) able to (deterministically) find a correct satisfying solution to the formula.

Note that the underlying group in terms of its representation, independent of the calculus (presentation), is always commutative. The truth assignments that compose any solution identified by the calculus can always be reordered in a different way by some other algorithm that satisfies the underlying (satisfiable) formula. Thus, the group is always abelian, independent of the calculus.

The identity of this group is important in defining the *relative* least fixed-point of our calculus as a logic — we build a topological closure for the solution space θ_{F_n} for a formula F_n with (arbitrarily many) n variables when we can *guarantee* such a group structure on all assignment paths.[9] The group identity, in addition to being valued as 0, is also the *least fixed-point* of our logic — which we can now conceive of arithmetically as being comprised of addition and multiplication of logical elements. This identity, being $\emptyset$, is also *clopen*, and is computed within the topological closure as a pairing with the boundary of the space.

Theorem 3.7. *A Transition Gadget G_i is an element of $R_i(T(+,\times))$, as well as $R_{i-1}(T(+,\times))$ if $i > 0$.*

Proof. Let $\{\{TDF\}_i\}$ be the set of all $\{TDF\}_i$ sets encountered at distinct points in time in some distinct differentiation trajectory so that $|\{\{TDF\}_i\}| = n$. Each G_i is a multiplication of half of the elements in $\{TDF\}_{i-1}$ *plus* G_{i-1} :

$$G_i = (\{TDF\}_{(i-1)_1} \times \cdots \times \{TDF\}_{(i-1)_{\frac{1}{2}|\{TDF\}_{(i-1)}|}}) + G_{i-1}.$$

Each G_i is also an addition of the remaining $\frac{1}{2}|\{TDF\}_{(i-1)}|$ elements in $\{TDF\}_i$.
Note that the addition operation of the additive group $\mathcal{G}$ of Transition Gadgets (of 2 vectors to get a third vector), is compatible *over time*, that, is over the path of the topological ring addition (see Example. 3.12), with the addition operation of $R_i(T(+,\times))$.
Thus, each G_i is an element of $R_i(T(+,\times))$, as well as $R_{i-1}(T(+,\times))$ if $i > 0$. □

Example 3.12. *Let C be a 3-SAT formula F with 1 clause C and with (as is the case with a formula with 1 clause) 8 possible solutions (counting the initially excluded unassignable statement at time $t = 0$) $\{F_1, \ldots, F_8\}$ and $\{G_1, G_2, G_3\}$ be the set of 3 transition gadgets that conduct the derivations at times t_1, t_2 and t_3. Let t_0 be the time at time $t = 0$ and G_0 be the integration gadget that computes the total solution space at t_0. Then the following arithmetic interactions hold :*

$$G_0 = F_1 + \cdots + F_8$$

$$G_1 = (F_1 \times \cdots \times F_4) + G_0 = F_5 + \cdots + F_8$$

$G_2 = (F_5 \times F_6) + G_1 = F_7 + F_8 = G_1 + G_0 \implies G_0 = F_5 \times F_6$ *by an inner product rescaling in the chosen direction*

$$G_3 = (F_7 \times 1) + G_2 = F_8 = G_2 + G_1 \implies G_1 = F_7 \text{ by an inner product rescaling}$$

where F_8 is the satisfying solution selected.

[9] This is a type of *motivic* cohomology which we hope to address in follow-up work.

Remark 3.19. *As we travel along the assignment trajectory, outer vectors get "rescaled" by a (multiplicative) conformal factor to correspond to the "divergence" at each point in time, so that the correct inner product is maintained at all times. The new length of each outer vector is a projection on to the updated direction of the identity/basis vector. The rescaling factor in this sense is the continuous Riemannian metric corresponding, via an ultraproduct mapping, to the normal discrete metric.*

Definition 3.42. *Define* $\bigcup_i R_i(T(+,\times)) \subseteq R(T(+,\times))$

Theorem 3.8. *The set of Transition Gadgets is an ideal of* $R(T(+,\times))$.

Proof. The set of Transition Gadgets $G \subseteq R(T(+,\times))$ is closed under addition and multiplication by any element $e \in R(T(+,\times)$ □

Theorem 3.9. *Ideals Are Powers of Predecessors .*

Proof. Between points in time, the preserved measure of the space is shared between an exponentially less number of elements in $R(T(+,\times))$, so that for ideals (which absorb the other elements in the indexed subring) I_i defined between points in time t and $t+1$, $I_{t+1} = (I_t)^2$ □

Theorem 3.10. *Each* $\{TDF\}_i$ *is defined as a quotient subring of the* $R_i(T(+,\times)$ *over which it is defined :* $\{TDF\}_i \leftarrow \{TDF\}_{i-1}/G_i$.

Proof. By corresponding to generators/recognizers, each $\{TDF\}_i$ is an element of the quotient computed by the indexed ideals, that is, transition gadgets. □

3.4.8 Algebra Of Differentiation — Modules

We can now frame the sequence D_n, of derivations that compose a solution to some formula F_n, by the additive group action of ideals (transition gadgets). D_n is a sum over variable assignments, of differentiations of a solution space — with *negative* constraint functions, constructed via the action of σ-functions introducing and eliminating generators/recognizers — as antiderivatives, parameterized by time — acting as coefficients to partial derivations of satisfying solutions to F_n.

Such a framing of D_n corresponds to an extension of Eqn. 3 (scoped to $\mathcal{A}(t)$) to all time steps of an algorithm $\mathcal{A}$:
Let F_n be a 3-SAT formula with n variables.
And a *DetAlg* be the dependent function, F, as expressed in Eqn. 3.
And generators as constraint functions (introduced and eliminated by σ-functions) be the complex coefficients, $g(t_i)$.
And t_i be a distinct point in time.
Then the following expresses a *total* differentiation over the solution space of F_n.

$$\sum_{i=0}^{n} \left(g(t_i),\ \frac{\partial^2 f(r,t_i)}{\partial t_i^2} \right) = 0, \qquad r \in \mathbb{R},\ g(t_i) \in \mathbb{C}. \tag{5}$$

A PDE analogous to a vanishing Holant reduction (Expr. 1). Derivations are *Pfaffian* in that any derivation at time t is a sum of derivations at all prior times $< t$:

$$\frac{\partial^2 f(r,t_k)}{\partial t_k^2} = \sum_{i=0}^{k-1} \left(g(t_i),\ \frac{\partial^2 f(r,t_i)}{\partial t_i^2} \right). \tag{6}$$

The equation for a total derivation (Eqn. 5) must equal the derivative of the $(n+1)^{th}$, that is total, function. The value of this sum must go to 0, as a derivative, because our (final) solution has no positive forward

motion (remaining) that can be executed.[10]

In addition to the standard abstract algebraic structures introduced in the last section, we now also extend the following structures:

- *Differential Module ($\mathcal{D}$-module)*: A module over a ring of differential operators.
- *Differentially Closed Field*: A field in which derivations are closed — that is, every possible solution to a differential equation with a solution in an extension of the field, is an element of the field itself.

The reader need not worry about any other technical details of these structures apart from the ones stated above. Our goal here, this being a computationally focused work, is to derive, dare we say, a simpler version of an operational amalgam of these structures:

Definition 3.43. Differentially Closed Differential Module (*DC-$\mathcal{D}$-module*)*: A module M is a DC-$\mathcal{D}$-module over a ring R if all the following hold:*

- *I is an ideal in R.*
- *I defines an additive group.*
- *$M = \bigcup_{k\in\mathbb{N}} I^k$.*
- *Let an extension of R, $\bar{R}$,be defined pointwise as $R \cup\ I^{k+l}, l \geq 1$.*
- *For any partial differential equation P, if a solution of P exists in an extension $\bar{R}$ of R, then it exists in M.*

Theorem 3.11. *The set of transition Gadgets G is a module over $R(T(+, \times))$.*

Proof. By the following argument:

- G is the additive, abelian, normal subgroup of the larger group composed of constraint functions (quotient group) and transition gadgets.
- As a vector space, the scalars of G are points in time. This implies that its scalar multiplication is distributive over the operations of addition between elements of the ring or module, and is compatible with the ring multiplication.[11]

G satisfies the definition of a module (over a ring). □

As before, one can infer that the submodules of this module are also modules over the respective time-indexed subrings.

Theorem 3.12. *The set of Transition Gadgets $\{G_i\}$ is a Differential Module.*

Proof. By the following argument:

- Each Transition Gadget G_i corresponds to $\delta(t)$-function and is thus, a module element over a differential operator.

[10] Viewed as the process of computing a product (of constraints), the stated equation functions to compute this product at each point in time, by summing over previous products. This is similar to the notion of an Operator Product Expansion used for Vertex Operator Algebras, which takes products of operators (like we do in our calculus) by summing over the operators themselves. Operators there are the representations of fields, that is, mappings of point to values in spacetime (also, just as in our calculus).

[11] This essentially means that the addition between elements of $R(T(+, \times))$ is time-ordered.

- A sequence of $\delta(t_i)$ functions, by indexed correspondence with $\{G_i\}$, and thus with $R(T(+,\times))$, form a ring defined by a homomorphism:

$$\phi : \delta(t_i) \to G_i, \text{ so that } : \phi(\delta(t)) \subseteq \{G_i\}.$$

- The set of transition Gadgets is thus, as a module, by definition, a *Differential Module.*

□

As before, we can infer that each subset of the set of Transition Gadgets is a Differential Module over the respective time-indexed subrings.

Theorem 3.13. *The set of Transition Gadgets $\{G_i\}$, is a DC-$\mathcal{D}$-module over a ring.*

Proof. $\{G_i\}$ satisfies our definition of a *DC-$\mathcal{D}$-module* over a ring since:

$$\text{for any } G_i, \quad i > 0, \quad I_i = (I_{i-1})^2, \quad G_i \in I_i, \quad G_{i-1} \in I_{i-1}.$$

Where $\{I_i\}$ are ideals.
G_i simply eliminates solutions/open sets from $R_{i-1}(T(+,\times))$ so that:
G_i is a sum over solutions that already exist (as part of a sum) in G_{i-1}.
Thus, each I_i satisfies the definition of a *DC-$\mathcal{D}$-module* over a ring $R_i(T(+,\times))$ so that:
$\{I_i\} \equiv G$ is a *DC-$\mathcal{D}$-module* over $R(T(+,\times))$ □

The algorithm $\mathcal{A}$ (proof system) which determines and orders the *DC-$\mathcal{D}$-module*, by bounding its identity element, is an algebra (over a ring) - defined briefly as a module equipped with a bilinear product. The bilinear product P comes from the product form (mapping to the same codomain) of the pair of functions (σ, δ), computed by the calculus. P binds the identity of the underlying module to a topological boundary, thus creating the needed topological closure. As an algebra which *presents* the *DC-$\mathcal{D}$-module*, the calculus may or may not be commutative (see Remark, 3.18). The *DC-$\mathcal{D}$-module* itself is always however, commutative with respect to its representation. (again, see Remark, 3.18).

3.4.9 Formal Definition/Invariants Of Differentiation

We can now state the "axioms" of differentiation of a small system according to our reverse mathematical scheme. Note that invariants of our calculus, in addition to ensuring that the calculus, extended to a large system, adheres to specific technical requirements, also function to state these requirements in an algebraic manner. This ensures that these requirements are directly implementable in a calculus over other problems in the class *NP*, without the need to first reduce these problems to 3-SAT. These invariants are thus meant to serve, as best as possible, as encoding-agnostic statements of the properties required of such a calculus, in order to ensure its algebraic equivalence to the one we present in this article.

First, we define related algebraic concepts:

Definition 3.44. Scheme*: Briefly, a (commutative) scheme can be defined as a ringed space that generalizes a variety by distinguishing multiplicities (different solutions mapped to the same value) and features a (commutative) ring for every open set. A ringed space being a family of (commutative) rings parameterized by open subsets of a topological space together with ring homomorphisms that play roles of restrictions.*

One can easily verify from this definition that the topological space over which we perform derivations is a ringed space.

Definition 3.45. Fibration*: A scheme is usually described (referred to as Grothendieck's relative point of view) as embedded within a diagonal morphism from a family of schemes X on to a particular scheme Y, expressed as: $X \times_Y X \longrightarrow X$. The set X is said to fiber over Y, the cotangent bundle on $X \times_Y X$ on the left of the arrow being the pullback and the tangent vector establishing X on the right being the pushforward along the fibers of Y.*

In our case, we do not have direct access to the (0^{th}-order) fiber products over the scheme, but have instead defined a (first-order) transition gadget over such a morphism. It is this gadget over which we have defined pullbacks and pushforwards. In other words, instead of working with the ring spectrum (set of all prime ideals) of *DetAlg* functions (called affine schemes in the literature), we have instead, embedded the morphisms of these schemes into the ring spectrum of a *DC-$\mathcal{D}$-module*.

Remark 3.20. *Note that a clause can be regarded as a family of schemes (solutions), for some arbitrary polynomial representation used in a proof system (such as a $\{0,1\}$-valued polynomial structure [34]), of $F(n)$. In this sense, the total solution space of any satisfiable formula can be regarded as encoded by a family of schemes.*

The following is a (unified) update, to the working Definitions. 3.13 and 3.15 of derivations (differentiation), which revises and subsumes these definitions. The definitions of derivatives (Definitions. 3.14 and 3.16) stay unchanged with reference to this updated definition.

Definition 3.46. Differentiation (small system)*: A differentiation Diff($\mathcal{F}_{\lambda(t_i)}$), of a DetAlg function $\mathcal{F}_{\lambda(t_i)}$ which satisfies a clause C at a point in time t_i, is:*

- *A morphism $\mathcal{M}$ of a differentiable family of schemes $\mathcal{S}$, where:*
- *$\mathcal{M}$ is embedded within the pushforward τ of a Transition Gadget G, $\tau : G_i \times_G G_i \longrightarrow G_i$, $\mathcal{M} \hookrightarrow \tau$.*
- *$\mathcal{B}$ is a DC-$\mathcal{D}$-module over a ring R and $G \in \mathcal{B}$.*
- *Solutions to the partial differential equation P, which embeds $\mathcal{S}$ in $\mathcal{B}$, by the group action of $\{G_i\}$, are closed in $\mathcal{B}$.*
- *$\mathcal{F}_{\lambda(t_i)}$ is a solution to P and $\mathcal{F}_{\lambda(t_i)} \in \mathcal{S}$.*
- *If $t_i < 3$, then:*
  - *Diff($\mathcal{F}_{\lambda(t_i)}$) $= \mathcal{F}_{\lambda(t_{i+1})}$*
  - *Diff($\mathcal{F}_{\lambda(t_i)}$) satisfies Definition. 3.13*
- *If $t_i = 3$, Diff($\mathcal{F}_{\lambda(t)}$) satisfies Definition. 3.15.*
- *If t_i is not time $t = 0$, then at all points in time t_j, $0 \leq j < i$, Diff($\mathcal{F}_{\lambda(t_j)}$) satisfies all the conditions listed before this one, recursively.*

Remark 3.21. ***Note*** *that we have not restricted the family of schemes $\mathcal{S}$ to only encode solutions for a single clause C, so that it seems that we can make this a definition for a formula with an arbitrary number of clauses if we remove the single clause condition along with the $t \leq 3$ implication. This is partly true, except that, so far, we can only ascertain that this definition works for a single clause, and have not yet introduced any additional conditions that will need to be met by a larger formula. We will keep this definition as is, for the sake of algorithmic modularity. This will enable us to subsume it in a straightforward manner, by an updated definition $\mathcal{D}$, by including its satisfaction as a step in $\mathcal{D}$, like we did for the working Definitions. 3.13 and 3.15 in this definition.*

Definition. 3.46 presents function precompositions (and compositions) of a function $f(x)$, of a *nilpotent* identity x, of a module M, by the pullbacks (and pushforwards) of products of fibers (fibration) of M. This provides a natural setting in which to define integration as the higher dimensional precompositional pullback of the complement of x. This definition is also complete for a small system, and places a (algebraic) structural restriction on possible extensions of the definition of differentiation for a large system (with arbitrarily many clauses and variables). Such a (larger) system must necessarily have the following properties:

Property 3.1. *Differentiation (as well as integration by definition of the calculus) must occur within a topological space.*

This property ensures that there is a topology which can be *closed*, as per our requirement for a meta-model as providing such a closure for the (regular) model of the calculus. This topology must necessarily be an ultraproduct construction from the topology of individual clauses as noted in Remark. 3.1.

Property 3.2. *The* topologically closed *space over which differentiation (and integration) occurs for a formula F must correspond pointwise (in time) with elements of a DC-$\mathcal{D}$-module over a ring.*

This connects the notion of a differential ring closure with that of topological closure and sets the preference for a nested system of such closures in correspondence with ideals.

We can add an even stronger requirement, if we think of codifferentiation as a functor, $\mathcal{F}$, precisely from the category $\mathbf{Cat}(\textit{Assignable Statements})$ to the set $\{\textit{Unassignable Statements}\}$.

$$\mathcal{F} : \mathbf{Cat}(\textit{Assignable Statements}) \longrightarrow \{\textit{Unassignable Statements}\}.$$

Definition 3.47. Chiral Object*: $\mathcal{F}$ is by definition, a presheaf on the set of Unassignable statements, that is, a functor from the opposite category to the underlying set:*

$$\mathcal{F} : \boldsymbol{Cat}^{op}(\textit{Unassignable Statements}) \longrightarrow \{\textit{Unassignable Statements}\}.$$

*$\mathcal{F}$ also corresponds with, and is embedded by, a set of transverse morphisms Γ between assignments, which form another category **Cat**(Assignments), with assignments as objects.*

***Cat**(Assignments) is defined (indexed) externally by a symmetric monoidal category **Cat**(Chiral), whose objects are composed of tensor products of these pair of objects:*

- *A universal object from **Cat**(Unassignable Statements).*
- *A local object from **Cat**(Assignable Statements).*

*We **define** these composite objects as higher dimensional Chiral Objects, computed by the pair (σ, λ) so that: $(\sigma \bigotimes \lambda) \longrightarrow assignment$.*

A *Chiral Object* at some point in time t, can be viewed as a representation of the *group* of all possible directions of motion at t. At time t, we compute a particular direction by projecting this representation in a particular direction (chirality) and treating the other (previously) possible directions as complementary directions of the selected one. The chosen direction is also (inductively) a *Chiral Object*.

Let $\mathcal{H}$ be the functor from $\mathbf{Cat}(\textit{Chiral})$ to $\mathbf{Cat}(\textit{Assignments})$:

$$\mathcal{H} : \mathbf{Cat}(\textit{Chiral}) \longrightarrow \mathbf{Cat}(\textit{Assignments}).$$

If we now think of differentiation also as a complementary functor $\mathcal{G}$, of $\mathcal{F}$, then there must exist a natural transformation between $\mathcal{G}$ and $\mathcal{H}$. And so, we can add the following property to the necessary requirements of a larger system of differentiation based on the above analysis:

Property 3.3. *Differentiation must be indexed by a Chiral Object.*

This requirement is finer-grained than the first two, in that it helps us specify not just the large-scale properties of the space, but also the smaller scale ones of the objects that constitute the space, as well as specify the functions that compute them.[12]

This notion of a chiral structure is extended from that of Chiral Algebras in 2-dimensional algebro-geometric spaces, originally introduced by Beilinson and Drinfeld [29]. The authors define chiral algebras

[12] Endowing this analysis with the notion of a *site* within a universe, and thus, a topos-theoretic reading.

for algebraic curves embedded within a $\mathcal{D}$-module $\mathcal{A}$, defined via a Lie bracket, with the following module homomorphism (subject to extra technical requirements), as:

$$\mu : j_* j^* (\mathcal{A} \boxtimes \mathcal{A}) \longrightarrow \Delta_! \mathcal{A}$$

Where the Δ are the inclusion of the complement of the diagonal (morphism) and the diagonal respectively and μ is required to be antisymmetric with regards to extra technical requirements that pertain to a satisfaction of a certain version of the Jacobi identity.[13]

3.5 Integration

Integration as a dual operation to differentiation will now also be reverse-defined.

3.5.1 Algorithmic Structure Of Integration

We can immediately infer that a 3-SAT formula with 1 clause has a well-defined derivative at time $t = 0$ as well as a well-defined antiderivative at time $t \geq 1$. This means that the domain and codomain of all subsequent σ and λ-functions, as well as all transition gadgets, is well-defined and can be computed. A 3-SAT formula with 1 clause is thus said to be a well-defined *system of* 3 *variables*. The concept of such a *system* is extendable to the more general case as:

Definition 3.48. A system of n variables *F(n), is defined as a system of differentiation and integration over the solution space* θ_{F_n} *of a* 3*-SAT formula* F_n *with* n *variables and arbitrarily many clauses.*

We can immediately prove the following about integration with respect to *F(n)*.

Proposition 3.8. *F(n), is integrated, with respect to a differentially closed field* Γ*, if* $\forall D$*, where* D *is a satisfying solution to F(n),* $\exists \mathcal{L}(t_i) \in \Gamma$*, where* $\mathcal{L}(t_i)$ *is an ordered sequence, that is, a lattice, of transition gadgets* t_i *at each point in time* i*,* $1 \leq i \leq n$*, that derive* D*.*

Proof. We prove by contradiction.
Assume $\exists D$ where $\nexists\ \mathcal{L}(\mathrm{t}_i)$. By Definition. 3.46, the λ-function which differentiates D cannot be computed with respect to Γ. By the usual definition of integration as dual to differentiation, *F(n)* is not integrated, with respect to Γ. □

We will use the implied lattice $\mathcal{L}(\mathrm{t}_i)$ to define a differential form closure below in Section. 3.5.5.

Proposition 3.9. *In a system of* n *variables with* 2 *clauses, F(n, 2) that we wish to satisfy, let the following hold:*

- *Let* C_1 *be the first clause and* C_2 *be the second clause.*
- *Let the* Unassignable Statement*,* U_1 *recorded for* C_1 *contain* ***at least AND no more than****,* 1*, that is, exactly just* 1*, truth assignment* ρ *which negates another truth assignment* $\neg\rho$ *in the* Unassignable Statement U_2 *recorded for* C_2*.*
- *Let* $\mathcal{R}_1$ *be the constraint function of* C_1 *and* $\mathcal{R}_2$ *be the constraint function of* C_2*.*

  For example, '1,2,3' and '1,2,-3' form such a (U_1, U_2) *pair and so does '1,2,-5' and '3,4,-5'. '1,2,3' and '1,-2,-3' however do not form such a pair on account of having* 2 *pairs of mutually conflicting assignments. '1,2,3' and '1,2,4' also do not form such a pair, having no conflicting assignment in common.*

[13] All conditions met by our presentation of the calculus so far, which can also be viewed as being given over a Lie bracket $[\sigma, \lambda]$, regarded as functions of vector fields (each module morphism being an individual vector space), with λ differentiating over the flow of the field generated by σ codifferentiations. The interested reader can further investigate the connections and similarities between our algorithmic presentation of the same notion and the full mathematical theory as laid out in [29], which is beyond the scope of, and not necessary for the rest of the presentation.

Then, we must create a new unassignable statement U_3, which records a conjunction of all the assignments recorded in ***BOTH*** *U_1 and U_2, excluding the contradicting assignments, and U_3 must be recorded to a constraint function $\mathcal{R}_3$ of a new clause C_3.*

> *For example, for '1,2,3' and '1,2,-3', we must create a new unassignable statement (represented by) '1,2'. For '1,2,-5' and '3,4,-5', we must create a new unassignable statement (represented by) '1,2,3,4'. In both cases, the new unassignable statement must be recorded to a new constraint function (if such a constraint function does not exist yet).*

Note *that $\mathcal{R}_3$ will represent a new clause, with a length that may be shorter, equal to or greater than the length of either $\mathcal{R}_1$ or $\mathcal{R}_2$.*

Proof. If we allow execution of the statement of U_3, *F(2)* cannot be satisfied. But we know (trivially) that *F(2)* is satisfiable and that the statement which we have recorded in U_3 is unassignable. Therefore, U_3 should be recorded to a new constraint function, namely $\mathcal{R}_3$ for a new clause C_3. □

Note Correspondence: That a reading of this purely as a propositional operation is a type of resolution [32, 33]. We however use it in the context of (time-ordered) differentiation, as pointed out for the case of unit propagation.

Such a system $F(2)$ as we have just described, is well-defined (with regards to both operations of the calculus). From this, we can infer the following rule of integration (verbatim from the above proposition):

Rule 3.4. *In a system of n variables with 2 clauses F(n, 2) that we wish to satisfy, let the following hold:*

- *Let C_1 be the first clause and C_2 be the second clause.*
- *Let the* Unassignable Statement*, U_1 recorded for C_1 contain* ***EXACTLY*** *one truth assignment ρ which negates another truth assignment $\neg\rho$ in the* Unassignable Statement *U_2 recorded for C_2.*
- *Let $\mathcal{R}_1$ be the constraint function of C_1 and $\mathcal{R}_2$ be the constraint function of C_2.*

Then: **Create** *a new unassignable statement U_3, which records a conjunction of all the assignments recorded in* ***BOTH*** *U_1 and U_2, excluding the contradicting assignments, and record U_3 to a constraint function $\mathcal{R}_3$ for a new clause C_3, if such a clause does not yet exist (to avoid duplication).*

A well-defined system of n variables with 2 clauses which implements this rule naturally satisfies the properties of differentiation that we defined earlier. It also provides a base case for the topological ultraproduct we noted in Remark. 3.1.

With this basic operational view of integration, we can go on to define an additional property with regards to the antiderivative computed by the integration for a system of n variables. Note that the antiderivative endows the category of differential objects (functions) with an initial limit, a boundary, which pairs with a least fixed-point, the identity of the *DC-$\mathcal{D}$-module*.

We define *abelian categories*, in the usual (simplified) sense used for externally sourced algebraic definitions in this paper:

Definition 3.49. Abelian Category*: Abelian categories are additive categories, where an additive category is a pre-additive category — a category whose morphisms correspond to a symmetric monoidal category of abelian groups — with all finite biproducts — meaning products, which correspond to satisfying solutions, and coproducts, which correspond to a topological boundary, as limits of the category.*[14]

If we enforce that any morphism of chiral objects has a coproduct, we obtain a proper antiderivative for every morphism, obtaining an abelian category by construction, since products are already expected for all morphisms, by contract of the calculus. We can *strengthen* Property. 3.3 for differentiation as follows:

[14] Again, we recommend the interested reader to begin more in-depth investigations with [35], since abelian categories as we have used them in this paper – relating sheaves of groups to ($\mathcal{D}$-)modules in the context of operators defined within a topological space — is an example of a Grothendieck Category (referred to as such in the mathematical literature) as introduced in that paper.

Property 3.4. *Differentiation must be indexed by a chiral object from an abelian category of chiral objects,* ***AbelianCat****(Chiral).*

3.5.2 Symmetric Differential Forms

We employ the notion of a *differential form*, briefly defined, as a structured way of *defining* integrands for different types of structures (curves, surfaces, solids, and higher-dimensional manifolds), to help organize and define the dualization between differentiation and integration.
Consider the following expression, defined for a system of n variables, $F(\mathrm{n})$:

$$\int \sum_{i=0}^{n} \bigwedge_{j=0}^{|Sch(x)|} f\left(x_1, ..., x_n\right) \, d_j x_i. \tag{7}$$

We define differential forms, as used by our calculus, step-wise, by breaking down this expression.
First we **define** the expression, *Sch(x)*, over which the $\wedge$ symbol (**wedge product**), as the family of *all* schemes that satisfy $F(\mathrm{n})$. Any particular solution, $S(\mathrm{n})$, is a *fibration* of *Sch(x)* over $S(\mathrm{n})$:

$$S(x_1.....x_n) := Sch\left(x\right) \times_{S(n)} Sch\left(x\right) \longrightarrow Sch\left(x\right).$$

Recall (Remark. 3.20) that *Sch(x)* is just an encoding of the solution space of all satisfying solutions. $|Sch(x)|$ is then the number of all satisfying solutions.
The following expression is the n-form:

$$\sum_{i=0}^{n} \bigwedge_{j=0}^{|Sch(x)|} f\left(x_1, ..., x_n\right) \, d_j x_i. \tag{8}$$

Note the similarities of this expression to the Holant (Expr. 1) as well, where in this case, we take a sum of wedge products.

Each d_j represents the differential (second-order partial differentials taken at any point in time) of a variable x_i, in a particular direction j (complete solution), of the ambient manifold (which embeds/charts all possible solutions), at time $t = 0$, and is as used for standard differential form denotation.

For example, if we expand out this expression for a *2*-form for a formula with 3 variables and 2 solutions, we get:

$$f\left(x_1, x_2, x_3\right) \, d_1 x_1 \wedge d_2 x_1 + f\left(x_1, x_2, x_3\right) \, d_1 x_2 \wedge d_2 x_2 + f\left(x_1, x_2, x_3\right) \, d_1 x_3 \wedge d_2 x_3. \tag{9}$$

If we view an assignment trajectory as a vector $\vec{v}$ in a vector space V (as expressed by a *DC-$\mathcal{D}$-module*), then for a *2*-form, the wedge product expresses the *bilinear* (and in general, n-linear for an n-form) *geometric product* of 2 vectors — v_1 and v_2 — each belonging to a distinct 1-dimensional subspace $V_{dim(1)}$ of V — as an element v_3 of a larger subspace $V_{dim(2)}$ of V, of dimension 2. v_3 is **defined** as the *wedge product* or *exterior product* of v_1 and v_2 and is expressed as: $v_3 = v_1 \wedge v_2$.[15]

We can compose exterior products linearly to form an exterior product of n-vectors in n-dimensional space, Such a product is referred to as an n-blade.

Exterior products are further defined by the alternating property that: $v_1 \wedge v_1 = 0$ and $v_1 \wedge v_2 = -v_2 \wedge v_1$. This "interior" property will be moot to us as we will be manipulating blades via their *exterior powers* –

[15]For the interested reader, note that our underlying *transverse* manifold M_T is fitted with a *symplectic 2*-form at every point in time which is compatible with its complex structure and also defines a *positive definite* Riemannian metric. This defines each ambient manifold M_A as a Kähler manifold and M_T as a hyper-Kähler manifold as used in Kähler geometry.

Sections 3.5.3 and 3.5.4.

Note that the way we have used and denoted n-forms is highly structured (symmetric) and modified to meet the needs of our calculus. For example, a *2*-form (Eqn . 9) in standard differential geometry can take the form:

$$f(x_1, x_2, x_3)\ dx_1 \wedge dx_2 + f(x_1, x_2, x_3)\ dx_2 \wedge dx_3 + f(x_1, x_2, x_3)\ dx_3 \wedge dx_1$$

The main idea being that we compute, pairwise, *2*-form wedges of different combinations of distinct differentials, for all possible areas described by the *2*-form[16]

As such, our differential form is *blade-symmetric* over any n number of variables and k number of possible solutions — with one blade per variable, and all blades having the same number of possible directions (each variable exists in each solution) at each point in time .

Definition 3.50. n-form function*: We will **define** every type of function f (λ, σ, $\delta(t)$) that corresponds with an n-form, an n-form f-function.*

For example, let a total derivation of a (complete and satisfying) solution to a satisfiable system of n variables be **described** as a *0*-form λ-function. Thus, a completed λ-function at time $t = n$ is a *0*-form λ-function. This **defines** a λ-function, $\lambda(t_0)$ for a system of n variables with p satisfying solutions, for which no differentiations have occurred, as a p-form λ-function.

Remark 3.22. *Recall that at time $t = 0$, $p = 2^n$ and that p is halved at every time step i and so at every i, corresponding to a truth assignment to a distinct variable (which may be grouped together with other assignments in a time block), the number of possible solutions, d, decreases exponentially so that $d = p/2^i$. For example, if $p = 16$, at time $t = 1$, $d = 16/2 = 8$ and at time $t = 2$, $d = 16/4 = 4$. Thus, at any time point in time $t_i = i$, there are:*

- *$k = n - i$ number of blades.*
- *The size (number of distinct differentials of a variable) of each blade is $2^k = p/2^i = 2^{n-i}$.*
- *The differential form is a (2^k)-form $\equiv$ $(p/2^i)$-form $\equiv$ (2^{n-i})-form and so each blade is a (2^k)-blade $\equiv$ $(p/2^i)$-blade $\equiv$ (2^{n-i})-blade.*

3.5.3 The Exterior Algebra

We first describe both the notions of an exterior power and an exterior algebra and then provide formal definitions for their *indexed* forms:

Exterior Power: Let ψ be the superset of all partial set of assignments in which the variable x_i, differentiated by the 1-dimensional vectors of a 2^k-blade ρ, is not assigned any truth value at time $t = n - k$.

For a system of n variables F(n), the linear span Θ, of ρ, is the set of all linearly independent vectors (sets of n satisfying assignments) that span (contain as a subset) ψ.

For example, $d_1x_1 \wedge d_2x_1$ in Expr. 9 with x_1 is a blade spanned by a set Θ with 2 solutions (and so is the other blade for x_2). Blade-symmetry ensures that all blades, at all times, will always have the same span. This span, scoped to each blade, is **defined** as the *exterior power* of the blade.

Exterior Algebra: If we make a distinction between the exterior powers of different blades of a vector space V by their scoping, then the direct sum of all the distinct exterior powers (which are simply copies of the same power) of V at time $t = n - k$ is **defined** as the *exterior algebra* at time t.

[16]Derivations as we have described them so far, are complex derivations (see Theorem. A.1 from the appendix). We leave it to the interested reader, after consulting the appendix, to verify that our differential form as we have outlined it, satisfies a definition of a complex (p, q) differential form.

Definition 3.51. $(2^k)^{th}$ Exterior Power*: Let V_k denote a 2^k-dimensional subspace of a vector space V of dimension 2^n. At time $t_i, t_i = i, i = n-k$, in a system of n variables F_n with DetAlg $\mathcal{D}$, the $(2^k)^{th}$ Exterior Power of a (2^k)-blade scoped to a variable v — is the set of internal nodes N:*

$$(N = \{\{0,1\} \to v\} \restriction_{\{V_k\}}) \ \wedge \ (\ell \coloneqq \{0 \oplus 1\}, \ell \in V_k \iff \ell \in \mathcal{D}).$$

That is, solutions, which can be transitioned to by the (2^k)-form $\delta(t)$-function computed at t_i.

Definition 3.52. $(2^k)^{th}$ Exterior Algebra*: At time $t_i, t_i = i, i = n-k$, in a system of n variables F_n, the* $(2^k)^{th}$ Exterior Algebra *is the direct sum of the $(2^k)^{th}$ exterior powers of all the (2^k)-blades.*

Theorem 3.14. *In a proof system $\mathcal{A}$, at time $t_i, t_i = i, i = n-k$, the $(2^k)^{th}$* Exterior Algebra Π *of a (2^k)-form $\delta(t)$-function $\mathcal{D}_i, 0 \leq i \leq n$ for $F(n)$ corresponds with the* antiderivative*, ρ, computed by $\mathcal{D}_i$ at t_i.*

Proof. At any time $i = n-k$, ρ, by being the coefficient of differentiation, corresponds with the set of all the remaining k number of assignments α, that $\mathcal{A}$ can make. The following statements then hold:

- α indexes the set β, of all (remaining) 2^k derivable sets of assignments, which when:
- β is scoped to each unassigned variable then:
- $\Pi \equiv \bigoplus_{j=1}^{X} \beta_j \times (1) = \beta, X = 2^k$.
  That is, Π is equivalent to —factoring out — and then direct summing — all the common elements (vectors) among all the exterior powers of the (2^k)-blades — leaving the identity 1 in the last bracket. This expression, the *symmetric* direct sum of exterior powers, then reduces to just β, which is represented in the space as a disjoint union/direct sum of each distinct solution/open set.

Since ρ via α indexes β at time t_i, Π via β corresponds with ρ. □

This allows us to describe integration for F(n) by connecting it to differentiation via a *geometric dualization* of the exterior algebra. First, we redefine $\delta(t)$ at any time t for F(n) in terms of the exterior algebra:

Definition 3.53. *A $\delta(t)$-function at time $t_i, t_i = i$ as computed by a pullback: A $\delta(t)$-function at time $t_i, t_i = i$ for a system of n variables F_n, corresponds with the $(2^{n-i})^{th}$ exterior algebra $\mathcal{E}$, of a 2^n-dimensional vector space V so that:*

- *Let $n - i = k$.*
- *β is the set of all remaining sets of truth assignments (satisfying or not) to unassigned variables in $F(n)$, $|\beta| = 2^k$.*
- *The $(2^k)^{th}$ dimensional subspace V_k of V has $k \times 2^k$ number of (2^k)-blades at time t.*
- *The* symmetric *direct sum of exterior powers of V_k with β scoped to each unassigned variable is:*

  $$\bigoplus_{j=1}^{X} \beta_j \times (1) = \beta, X = 2^k.$$

  Which corresponds to the direct sum of derivable sets of assignments, computed by $\mathcal{E}$.
- ***Then** $\mathcal{E}$ computes the pullback $\mathcal{P}$ of the $\delta(t)$-function, where $\mathcal{P}$ is a cotangent bundle mapping the $\delta(t)$-function (as a sheaf) to the topological boundary B of the solution space so that:*

  $$\beta \equiv \mathcal{P} \cong T(\delta(t))^* B.$$

$\implies$ *that each transition gadget G_i corresponding to a pulled back $((2^k))$-form $\delta(t)$-function, computes the integration of the $((2^k))$-form, which we can now infer, as being a DetAlg function, in the form of Expr. 7.*

The above implies that the proof system, $\mathcal{A}$, functions as an exterior algebra w.r.t to integration.

3.5.4 Formal Definition — Integration

Definition 3.54. Integration (small system) *An integration Int($\mathcal{F}_{\lambda(t_i)}$), **is a function** of a DetAlg function $\mathcal{F}_{\lambda(t_i)}$ for F(n) at time $t_i, t_i = i$, defined as follows:*

- *Int($\mathcal{F}_{\lambda(t_i)}$) is a morphism $\mathcal{M}$ of an integrable family of schemes $\mathcal{S}$.*
- *$\mathcal{F}_{\lambda(t_i)} \in S$.*
- *$\mathcal{M}$ is embedded within the pullback τ of a Transition Gadget G, as a $\delta(t)$-function. $\tau : G_i \times_G G_i \longrightarrow G_i$ $\mathcal{M} \hookrightarrow \tau$.*
- *$\tau \coloneqq \bigoplus_{j=1}^{X} \beta_j \times (1), k = n - i, X = 2^k$.*
- *Int($\mathcal{F}_{\lambda(t_i)}$) satisfies Rule. 3.4 for any pair of clauses C_x and C_y, $x \neq y$, in F(n).*
- *If t_i is not time $t = 0$, then:*
  - *Int($\mathcal{F}_{\lambda(t_i)}$) = $\mathcal{F}_{\lambda(t_{i-1})}$.*
  - *At all points in time t_k, $0 \leq k < i$, Int($\mathcal{F}_{\lambda(t_k)}$) satisfies all the conditions listed before this one, recursively.*

Remark 3.23. ***Note*** *that we have chosen to define integration for all F(n) and not just F(n, 2). This is because, like the definition of differentiation, we intend to subsume this definition under an updated definition $\mathcal{D}$ but unlike the definition of the differentiation operator, this definition is dependent on the number n of the variables in F(n). The way it is stated works for F(n, 2) (as well as F(3, 1)) where n can be any number, and also allows us to subsume it in a straightforward manner, as we will see, in $\mathcal{D}$. This definition contains the necessary modularity detail because integration as we use it in our calculus, works in an "exterior" way in relation to the clauses in F(n) (as exemplified by the use of the exterior algebra in this definition) versus the interiority of differentiation. This definition also serves a more fundamental purpose as an inductive base case (also stemming from this so-called exteriority property) for defining the ultraproduct construction used to complete the calculus.*

Thus, integration, at time $t = 0$ computes the *prime ideal* of the *DC-$\mathcal{D}$-module* M, under which we differentiate the solution space of F(n). It also implies that differentiation is obtained via a power system of closed ideals comprising M, which as expressed earlier, is the strongest form of differential closure. As noted, this also operationally defines integration as a (function compositional) pullback of the (additive) identity of M.

3.5.5 Formal Definition — Differential Form Closure

We now have the framework over which we can define the formal but simple notion of a *differential form closure* for F(n), as an ordering structure.

Definition 3.55. Differential Form Closure*: A* Differential Form Closure *on F(n) is a structure Φ in which:*

- *Every satisfying solution $d \in Sch(x)$ to F(n) is derivable as a vector $\vec{v}$.*
- *Each d has an antiderivation a_{t_i} at each point in time t_i, $1 \leq i \leq n$, computed by an integration with Definition. 3.54.*
- *Each t_i corresponds to a point on a lattice of transitions $\mathcal{L}(t_i)$ so that $t_i < t_{i+1}$.*

This definition implies that, at every point in time, we compute a (functor) sheaf into the direct sum/disjoint union of differential forms, dual to a twisted (many-valued) differential structure ($\mathcal{D}-module$), without burdening the definition with the technical details of n-blades, modules or a differential form. As such, Definition. 3.5.5 is agnostic to the mathematical details of how Φ is represented and has the following advantages:

- **Algebraic Simplicity**: The definition is similar to that of an algebraic closure as it focuses on solutions (roots).
- **Nullstellensatz Implication**: Antiderivative-derivative chaining implies a progression of powers (exponentially decreasing subsets) of elements of the structure, and as such, the Strong Nullstellensatz.
- **Algorithm Friendly**: One can speak of a differential form closure as being computed by an algorithm, without burdening the description with the underlying algebraic details. Differential form closures as we will compute them by the calculus can have a generic representation as a **set of unassignable statements** over which an algorithm orders the underlying vector space and lattice structures.

4 Induction Strategy

To induce the definitions we have secured so far on to a larger system, we will follow a slightly enhanced version of our reverse mathematics strategy, which has the following high-level features for a system of n variables, F(n):

- **Completable Algorithm**: We start with an algorithm, which is intended to be a proof system, which may not be complete and fill in the missing steps as we encounter them within the scope of theorem proving. We refer to this as a fill-in-the-gaps induction sub-strategy.
- **Limit Computation**: As per our declaration of polynomial time computability, we seek a fixed bound on the number of additional elements that will need to be added to a topological closure *TClosure*, of the solution space $\theta_{F(n)}$, of the (starting) set of clauses in F(n). Thus, we exhaustively enumerate conditions under which all starting 3-SAT clauses can be satisfied, up to some limit L.
- **Integro-Differential Closure Completion**: We require that the limit L expressed by *TClosure* respect the usual axiomatic interpretation of differentiation and integration by corresponding to a differential form closure ϕ.

These features will be concretely implemented by the supporting sub-strategies described in the following sub-sections:

4.1 Algorithmic Terminology

4.1.1 Stating Assignments Predicatively

We choose to use the terminology of ***assignment*** and ***unassignable statements***, since these terms are predicative in a way that makes an algorithm described using them correspond to its algebraically interpreted form as needed — all antiderivatives/coefficients of differentiation map to unassignable statements at limit (second-order) points as we will demonstrate, while assignments always map to (both possible and impossible) derivatives. Both are algorithmically testable for some required property of a truth predicate, and hence, (first-order) model of assignability, at any point in the assignment trajectory.

4.1.2 Generic Assignment Notation

Assignment notation convention is updated to reflect the genericity of the algorithm. The new notation will use bolded, unquoted and undelimited sequences of small letters instead of integers, and so, instead of having representations of the sort '1,20,-34' for an unassignable statement recorded to the constraint function of a clause and '-1' denoting an assignment, we will have generic representations of unassignable statements of the form **ab-c** and **a** (used for both an unassignable statement with only (conjunction of) one assignment **a** as well as an assignment **a** to some unspecified variable).

4.2 Blocking Scenarios

Definition 4.1. Blocking Assignment *: We define a* Blocking Assignment β *to a variable* x_i *in* $F(n)$ *by an algorithm* $\mathcal{A}$ *at time* t*, as :* $(x_i \coloneqq \beta) \implies \exists C_1, C_2 \in F(n), \exists x_j, i \neq j, \quad ((x_j \coloneqq 1 \implies C_1 \coloneqq 0) \wedge (x_j \coloneqq 0 \implies C_2 \coloneqq 0))$*, where* x_i *and* x_j *are variables in* $F(n)$*.*
that is, as a truth assignment by $\mathcal{A}$ *to some variable* x_i *which results in another variable* x_j*,* $i \neq j$*, not being assignable to either* 1*(true) or* 0*(false), because in either case, some clause in the underlying* 3*-SAT formula* will *become unsatisfiable.*

We identify 4 basic *blocking scenarios* for a system of n variables, F(n),

> Let ***Unassignable(t, ρ_1, ..., ρ_k)***, $1 \leq k \leq n$ be an ***n***-ary predicate that, at some point in time t, for any number of arguments, ρ_1, ..., ρ_k, returns ***True*** if all the arguments passed to ***Unassignable*** exist as unassignable statements in the differential form closure (fixed set of unassignable statements) of F(n) at t, and ***False*** otherwise.
> Let **x, y, z** and **i** be positive variable assignments and **-x, -y, -z** and **-i** be negative assignments to 4 different variables in F**(n)**.

At any point in time, t, for a derivation/differentiation performed at t on the solution space of F(n), the following blocking scenarios are defined :

> **Note** : Each sentence states in order — An ***Unassignable*** predicate which always returns true for the *set* of unassignable statements listed as its arguments — followed by the assignments forced by the scenario (if any, so this sentence may be empty/not exist) — followed by the stated assignment by the algorithm — and then, the resulting scenario.

Blocking Scenario 1 : *Unassignable*(**x**, **-x**). The algorithm **assigns** either **x** OR **-x**. The other clause becomes **unsatisfiable**.

Blocking Scenario 2 : *Unassignable*(**xy**, **x-y**). The algorithm **assigns x**. Both clauses become **unsatisfiable**.

Blocking Scenario 3 : *Unassignable*(**xy**, **x-z**, **z-y**). The algorithm **assigns x**. This **forces** an assignment to **z** (second clause). First and third clauses both become **unsatisfiable**.

Blocking Scenario 4 : *Unassignable*(**x-z**, **xi**, **z-y**, **y-i**). The algorithm **assigns x**. This **forces** assignments to both **z** (first clause) and **y** (third clause). The second and fourth clauses both become **unsatisfiable**.

The reader should study these scenarios carefully, both for understanding the notation use as well as why the scenarios are indeed blocking. This understanding will be crucial for following the rest of the paper, particularly the proofs of the induction theorems.
The strategy we will be using for ***preventing*** blocking scenarios is the employment of *witness statements.*

4.3 Witnessing

Witnessing as an element of our induction strategy consists of the following components:

4.3.1 Witness Statements

Definition 4.2. Witness Statement *: A* Witness Statement *is a statement, at some time* t *in the execution of a proof system* $\mathcal{A}$*, that an assignment to a variable* v *in* $F(n)$ *must be bound to a particular value* a*:* $\mathcal{A}(t) \iff v \coloneqq a$ *, as a result of applying Rule. 3.3 (negating an unassignable statement* $U, |U| = 1$ *by an opposite assignment which satisfies the clause).*

We briefly revert to our old assignment notation for the following example which demonstrates a witness statement.

Example 4.1. *Let U be an unassignable statement recorded to constraint function F of some clause C in the differential form closure of $F(n)$: $U \coloneqq$ '$11, 14, -17, 40, -50$' — and at some point in time, t, the proof system has selected the following assignments :* '$11, 14, -17, -50$'*, then '-40' is a witness statement since it states the assignment that satisfies the underlying clause.*

Witness statements are valued for their contribution to the algorithm : that their presence will be used to identify and prevent blocking scenarios. This is the first core element of our *fill-in-the-gaps* sub-strategy. We will refer to a witness statement as being made by a *witness*, defined as:

Definition 4.3. Witness *: A* Witness *in a proof system $\mathcal{A}$ is defined as an abstract entity able to make a* Witness Statement *at some time-step $\mathcal{A}(t)$.*

A witness always corresponds to a distinct clause with a recorded unassignable statement U contained in the differential form closure ϕ. All such clauses are associated with ϕ, but only the original set of clauses in the 3-SAT formula to be solved, are part of F(n).

4.3.2 Witness Depth

Used in the manner described above, witness statements, as well as free assignments (Definition. 3.6), function as the set of distributed "circuits" that compute a *DetAlg*, and so it is applicable to speak of a witness as having a witness depth. We first establish the following terms used to define a witness depth:

Definition 4.4. Auto-Differentiation*:* Auto-Differentiation *of a truth assignment ψ for a variable v in $F(n)$ is the forcing of the selection of ψ* as part *of computing the differential form/topological closure of $F(n)$ such that for any solution S_i of $F(n)$, $S_i \subseteq \theta_{F_n} \iff \psi \in S_i$ where θ_{F_n} is the solution space of $F(n)$. This implies that* every *solution/assignment set to $F(n)$ must contain ψ as a compulsory truth assignment to v because ψ is a global Bnd-Val (globally bound assignment).*

Note that our use of the term *auto-differentiation* is different from that meant by the term *automatic differentiation* used to describe work on finding derivatives of algorithms as described earlier.

Definition 4.5. Time $t = 0$ Of Differentiation*: We define $t = 0$ of differentiation for F(n) as the time* after *the proof system $\mathcal{A}$ has computed the differential form closure, that is, the total solution space of F(n) is integrated, but before $\mathcal{A}$ makes the first Free-Val (free assignment) selection.*

Remark 4.1. ***Note*** *that differentiation, in this sense, excludes auto-differentiation. This implies that the first assignment made at time $t = 1$ of differentiation is always a free assignment.* ***Note*** *also that this makes time $t = 0$ of differentiation distinct from the regular notion of time $t = 0$ (for a DetAlg, whose definition equates time instances with the total number of all assignable variables regardless of whether their assignments are auto-differentiated or normally differentiated).*

Definition 4.6. Witness Depth*: We define the* Witness Depth *of a witness as the number of variables contained in its unassignable statement at time $t = 0$* of differentiation*.*

For example (back to our generic notation), if, at time $t = 0$ of differentiation, the unassignable statement of a witness W is **ab-cd-e**, then the witness depth of W is 5.

4.3.3 Witness Boundary

If our algorithm is to be polynomial in its runtime as originally promised, then it is clear that the maximum depth of necessary witnesses has to be bounded by some constant d, say 10 for any system of n variables – then, we say that the proof system has witness boundary 10 or that the witness boundary of the system is of *depth*-10,.

4.3.4 Witness Correlation

Witness correlation is the process of *integrating* each witness W_i (corresponding to a clause in the differential form closure), pairwise with every other applicable witness $W_j, i \neq j$ a witness of a clause whose underlying unassignable statement shares a *single* conflicting assignment with the clause associated to W_i (as stated in Definition. 3.54 for a system of n variables with 2 clauses, and enforced by Rule. 3.4). Membership in {W}, the complete set of applicable witnesses, is computed according to a predicate defined by a specified witness boundary.

Algorithmically, witness correlation aims to compute the differential form closure of the satisfying solution space of F(n), by ***exhaustively*** integrating the solution spaces (corresponding to and represented by unassignable statements) of clauses in the closure, piecewise (2 at a time), adding new necessary witnesses (corresponding to clauses) indicated by each integration — up to the witness boundary. The witness boundary functions as an ultralimit needed for the topological ultraproduct construction noted in Remark. 3.1 (now also functioning as a model-theoretic/logical ultraproduct). Witness correlation for a satisfiable F(n) is complete when no new eligible witnesses, with depth not exceeding the defined limit, can be added to the closure.

5 Induction To F(n)

Here we develop theorems that implement an induction strategy comprising of the sub-strategies outlined in the preceding section for a system of n variables, F(n), to complete our definitions of the operators for F(n), that is, the large system.

To support the main inductive theorems, we develop supporting lemmas for inspecting a proof system at key points during the construction of a proof. For this we express the proof system as an algorithm $\mathcal{A}$ for which we present fragments of $\mathcal{A}$, scoped to some specific assignment scenario *Sc*. Each *Sc* is intended to **describe** one or more cases in which we may encounter a blocking scenario *B*, and then **prescribe**, where applicable, the necessary witness type (type corresponding to depth) needed to prevent *B*. The collection of all scenarios examined {Sc} should function as a witness boundary detection/definition procedure.
The logic expressed by each fragment of $\mathcal{A}$ examined will thus be of 2 kinds:

Complete Logic Fragments of this type have the property that witnesses with the witness type (depth) needed to make the necessary witness statements they imply already exist, with unassignable statements as part of the differential form closure ϕ. These kinds of fragments apply mainly to the more trivial theorems or to theorems following another theorem in which the witness type needed has already been established.

Completable Logic Fragments of this type have the property that witnesses with the depth of the witness type *D*, needed to make the necessary witness statements implied by the motivating scenario *Sc*, need to have their unassignable statements included in the closure computed by $\mathcal{A}$, in accordance with our fill-in-the-gaps sub-strategy. As per our contract, the maximal *D* indicated as necessary by {Sc} will be incorporated into the operational definition (of the witness boundary) of the differential form closure computed by $\mathcal{A}$ and thus used for a definition of a valid proof system.

We identify the 3 main types of witnesses we will use, and which will be defined as they are introduced in the text as:

- **Immediate Witnesses**: of which there are two sub-types, direct and indirect.
- **Intermediate Witnesses**: which are always indirect.
- **Boundary Witnesses**: which are always indirect.

A key component of determining the witness boundary will be the recognition of the fact that the differentiation portion of $\mathcal{A}$ need only consider clauses with witnesses of a certain type when inferring (implicitly) the total solution space of F(n). This defines the differentiation scope, as follows:

Definition 5.1. Differentiation Scope *: The* Differentiation Scope *of $\mathcal{A}$, and thus its fragments, is the set of satisfying solutions to the underlying* 3*-SAT formula, F. As such, the algorithm will only enforce the rules of differentiation (as given by Rules. 3.1 through 3.3) for the elements of the set of unassignable statements $\{U\}$, which we term the differential core, DiffCore, in the differential form closure that have a witness depth of* 3 *or smaller, that is, $|U| \leq 3$.*

The witnesses mapped to *DiffCore* is the set of ***immediate witnesses***.

We divide up the lemmas into sections, with each section dealing with a different type of immediate witness. These types will be identified with the witness depth at time $t = 0$ of differentiation, as defined. The last section will address theorems of mixed type. Each theorem will indicate the particular Blocking Scenario it addresses.

5.1 Witness Depth 1

Lemma 5.1. *A differential form closure ϕ (at time $t = 0$ of differentiation) cannot contain* Blocking Scenario 1.

Proof. By the definition of witness correlation for some formula F, which is the application of the integration rule. 3.4 over witnesses of F. If at any point in time, we resolve 2 witnesses in ϕ and obtain *Blocking Scenario 1*, regardless of whatever witness boundary we choose, then F is unsatisfiable. Differential form closures only apply to satisfiable formulas by definition. Therefore, a differential form closure cannot contain *Blocking Scenario 1.* □

5.2 Witness Depth 2

Lemma 5.2. *A differential form closure ϕ cannot contain* Blocking Scenario 2.

Proof. Witness correlation would combine both clauses **xy** and **x-y** into a single unassignable statement **x** (making **x** unassignable). □

Lemma 5.3. *A differential form closure ϕ cannot contain* Blocking Scenario 3.

Proof. Witness correlation would combine **xy** and **z-y** into **xz** which would then combine with **x-z** to make **x** unassignable.. □

Lemma 5.4. *A differential form closure ϕ cannot contain* Blocking Scenario 4.

Proof. Witness correlation would combine **xi** and **y-i** into **xy** which would then combine with **z-y** to form **xz** which would then combine with **x-z** to make **x** unassignable. □

5.3 Witness Depth 3

One can infer that differentiation of the solution spaces of clauses corresponding to witnesses of this type, via the *rewriting/updating* of their unassignable statements, can always result in any of the blocking scenarios, at some point time t in an assignment trajectory, *Traj*, traced out by $\mathcal{A}$.

Here, we address Blocking Scenarios $2, 3$ and 4 as they may be encountered in the course of plotting out *Traj*. *Traj* may encounter these scenarios because the *set* of associated clauses of witnesses of depth 3, $\{\Gamma\}$ — which have an unassignable statement U which records an assignment v, in conjunction with other assignments — are not satisfied by an assignment v selected by $\mathcal{A}$ at time t. A witness W for $C \in \{\Gamma\}$ **must** therefore have its underlying unassignable statement U rewritten by $\mathcal{A}$ into a reduced unassignable statement $U^{'}$ of depth 2. By addressing Scenarios $2, 3$ and 4, we are able to assess how these rewrites affect the continued satisfiability of the system, after the selection of v by $\mathcal{A}$.

The main proof strategy we will be using is: to demonstrate how we remediate each blocking scenario identified — by applying a witness statement S_W from some witness W (immediate or non-immediate as

applicable) — where $S_W, |S_W| = 1$, is deduced from unassignable statement U (included in the differential form closure ϕ) corresponding to W — and where S_W allows us to assign the negation $\neg\mathbf{x}$ of the blocking assignment $\mathbf{x}$, ***immediately*** it becomes unassignable. Where such a statement S_W comes from a witness W of depth d and where unassignable statements of witnesses of type *depth*-d are not currently included as part of the closure, we will incorporate the unassignable statements from witnesses of type *depth*-d going forward, into the closure, as we have noted.

In addition, the proof system (must) also ensure that such a forced assignment *correlates* with any other possible forced assignment. We achieve this in two steps:

- In contrast to the lemmas stated for the other 2 (simpler witness types), for this witness type, we can only state lemmas about the behavior of the algorithm at a distinct point in time t considered separately. Thus, we will qualify each lemma by making the assumption that up until time t, $\mathcal{A}$ has not encountered an inductive condition, namely, a blocking scenario.
- The global behavior of $\mathcal{A}$ over all time steps will be proved in Theorem 5.3 when we establish the absolute boundary of the the differential form closure.

Because of the complexity that comes with this technique, proofs will be broken down into applicable cases. In each presented case, we will use small letters of the alphabet $\notin S = \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{i}\}$ to denote the affected $\{\boldsymbol{v}\}$ assignment(s) for a particular clause C/set of clauses {C}. This enables us to reuse elements of S with the same connotation with which they were associated when we defined the resulting blocking scenario, for each theorem.
Note: Recall that all *assignments* are (intended to be) made by a *correct* (sound and complete) proof system $\mathcal{A}$ for a system of n variables F(n).

Lemma 5.5. *If $\mathcal{A}$ has not encountered a blocking scenario at any point in time $t_h, h < i \leq n$, then at time $t_i, t_h < t_i \leq t_n$, a free assignment to a variable in $F(n)$ will not result in* Blocking Scenario 2.

Proof. By proof of the following cases:

Case 5.3.1. *: Let **axy** and **ax-y** be two unassignable statements in the differential form closure of the solution space of F.*
*Then the assignment **a** will not lead to* Blocking Scenario 3 *by the following reasoning:*
*Witness correlation will combine both statements into **ax**.*
*Thus, we have an immediate witness that makes **x** unassignable once we have assigned **a**.*

Case 5.3.2. *: Let **axy** and **bx-y** be two unassignable statements in the differential form closure of the solution space of F.*
*Then the assignments **a** and **b** will not lead to* Blocking Scenario 3 *by the following reasoning:*
*Witness correlation will combine both statements into **abx**.*
*Thus, we have an immediate witness that makes **x** unassignable once we have assigned **a** and **b**.*

□

Remark 5.1. *: As one can readily observe in the last case demonstrated above, we can always switch the clauses in which two assignments, a and b, appear, to include one more case. But since we are using anonymous (letter) variables, this would amount to an abstract repetition of the same scenario, just as shown.*

As the scenarios get more complex, for cases that address scenarios which involve 2 *or more variables spanning* 3 *or more clauses, we will adopt an economical approach in listing down the complete list of applicable scenarios. Our approach can be summarized as follows : where assignments appear just once in some unassignable statement, the letters we use to denote each assignment will not matter, just like with the case above. In other cases, where the same assignment appears in more than one unassignable statement, this approach will need to be replaced with a more sophisticated and distinguishing one, as complete for the case. The basic approach in such cases where an assignment occurs* 2 *or more, say, x, times, among n number of clauses, is to "permute" the appearance of the assignment into $\binom{n}{x}$ number of cases, as will be clarified when we examine individual cases.*

Lemma 5.6. *If $\mathcal{A}$ has not encountered a blocking scenario at any point in time $t_h, h < i \leq n$, then at time $t_i, t_h < t_i \leq t_n$, a free assignment to a variable in $F(n)$ will not result in* Blocking Scenario 3.

Proof. By proof of the following cases:

Case 5.3.3. *: Let* $\boldsymbol{axy}$, $\boldsymbol{az\text{-}y}$ *and* $\boldsymbol{ax\text{-}z}$ *be three unassignable statements in the differential form closure of the solution space of F.*
Then the assignment $\boldsymbol{a}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{axy}$ *and* $\boldsymbol{az\text{-}y}$ *into* $\boldsymbol{axz}$ *which would then combine with* $\boldsymbol{ax\text{-}z}$ *to give* $\boldsymbol{ax}$ *and so, once* $\boldsymbol{a}$ *is assigned, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*.*

Case 5.3.4. *: Let* $\boldsymbol{axy}$, $\boldsymbol{az\text{-}y}$ *and* $\boldsymbol{bx\text{-}z}$ *be three unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{axy}$ *and* $\boldsymbol{az\text{-}y}$ *to give* $\boldsymbol{axz}$ *which combines with* $\boldsymbol{bx\text{-}z}$ *to give* $\boldsymbol{abx}$*, and so, once we assign* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*.*

Case 5.3.5. *: Let* $\boldsymbol{axy}$, $\boldsymbol{bz\text{-}y}$ *and* $\boldsymbol{ax\text{-}z}$ *be three unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{axy}$ *and* $\boldsymbol{bz\text{-}y}$ *into* $\boldsymbol{abxz}$*, a depth-*4 intermediate, indirect witness *(as it will not be directly differentiated) which combines with* $\boldsymbol{ax\text{-}z}$ *to give* $\boldsymbol{abx}$*, and so, once we assign* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*.*

Remark 5.2. *We introduced a depth-*4 intermediate witness *in this case. As this is a fill-in-the-gaps inductive theorem proving procedure, as per our reverse mathematical strategy, we assume that our differential form closure* must *now include unassignable statements for depth-*4 indirect witnesses*.*
We will formalize this assumption into a fact, when we define integration for a system of n variables in Section 6, in case we discover that we need even higher-depth witnesses to compute the closure. We will also carry this assumption implicitly into all cases that follow after the current one.

Case 5.3.6. *: Let* $\boldsymbol{bxy}$, $\boldsymbol{az\text{-}y}$ *and* $\boldsymbol{ax\text{-}z}$ *be three unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{bxy}$ *and* $\boldsymbol{az\text{-}y}$ *into* $\boldsymbol{abxz}$ *(a depth-*4 indirect witness *which combines with* $\boldsymbol{ax\text{-}z}$ *to give* $\boldsymbol{abx}$*, and so, once we assign* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*.*

Case 5.3.7. *: Let* $\boldsymbol{axy}$, $\boldsymbol{bz\text{-}y}$ *and* $\boldsymbol{cx\text{-}z}$ *be three unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{axy}$ *and* $\boldsymbol{bz\text{-}y}$ *to give* $\boldsymbol{abxz}$ *which combines with* $\boldsymbol{cx\text{-}z}$ *to give* $\boldsymbol{abcx}$*, and so, once we assign* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*.*

Remark 5.3. *The list of cases proven above entails the maximal context of any set of assignments spanning* 3 *different clauses which the theorem addresses.*

□

Lemma 5.7. *If $\mathcal{A}$ has not encountered a blocking scenario at any point in time $t_h, h < i \leq n$, then at time $t_i, t_h < t_i \leq t_n$, a free assignment to a variable in $F(n)$ will not result in* Blocking Scenario 4.

Proof. By proof of the following cases:

Case 5.3.8. *: Let* ***ax-z***, ***axi***, ***az-y*** *and* ***ay-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***ax-z*** *and* ***az-y*** *into* ***ax-y*** *which would then combine with* ***ay-i*** *to give* ***axi*** *which then combines with* ***ay-i*** *to give* ***ax****, and so, once we assign* ***a****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a****.*

Case 5.3.9. *: Let* ***ax-z***, ***axi***, ***az-y*** *and* ***by-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *and* ***b*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***az-y*** *and* ***by-i*** *into* ***abz–i*** *which would then combine with* ***ax-z*** *to give* ***abx-i*** *which then combines with* ***axi*** *to give* ***abx****, and so, once we assign* ***a*** *and* ***b****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a*** *and* ***b****.*
Note: *We could also have considered a different combination sequence (and indeed we can, for many of the cases listed):* ***axi*** *combines* ***by-i*** *to give* ***abxy*** *which combines with* ***az-y*** *to give* ***abxz*** *which combines with* ***ax-z*** *to give* ***abx****, and so, once we assign* ***a*** *and* ***b****, we cannot assign* ***x****. This shows that integration is in some sense "commutative". We will assume that this is true for subsequent proofs and will demonstrate only one such possible path in any proof.*

Case 5.3.10. *: Let* ***ax-z***, ***axi***, ***bz-y*** *and* ***ay-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *and* ***b*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***ax-z*** *and* ***bz-y*** *into* ***abx-y*** *which would then combine with* ***ay-i*** *to give* ***abx-i*** *which then combines with* ***axi*** *to give* ***abx*** *so that after assigning* ***a*** *and* ***b****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a*** *and* ***b****.*

Case 5.3.11. *: Let* ***ax-z***, ***bxi***, ***az-y*** *and* ***ay-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *and* ***b*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***ax-z*** *and* ***az-y*** *into* ***ax-y*** *which would then combine with* ***ay-i*** *to give* ***ax-i*** *which then combines with* ***bxi*** *to give* ***abx****, so that after assigning* ***a*** *and* ***b****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a*** *and* ***b****.*

Case 5.3.12. *: Let* ***bx-z***, ***axi***, ***az-y*** *and* ***ay-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *and* ***b*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***bx-z*** *and* ***az-y*** *into* ***abx-y*** *which would then combine with* ***ay-i*** *to give* ***abx-i*** *which then combines with* ***axi*** *to give* ***abx****, so that after assigning* ***a*** *and* ***b****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a*** *and* ***b****.*

Case 5.3.13. *: Let* ***ax-z***, ***axi***, ***bz-y*** *and* ***by-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *and* ***b*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***ax-z*** *and* ***bz-y*** *into* ***abx-y*** *which would then combine with* ***by-i*** *to give* ***abx-i*** *which then combines with* ***axi*** *to give* ***abx****, so that after assigning* ***a*** *and* ***b****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a*** *and* ***b****.*

Case 5.3.14. *: Let* ***bx-z***, ***axi***, ***bz-y*** *and* ***ay-i*** *be four unassignable statements in the differential form closure of the solution space of F.*
Then the assignment of ***a*** *and* ***b*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***bx-z*** *and* ***bz-y*** *into* ***bx-y*** *which would then combine with* ***ay-i*** *to give* ***abx-i*** *which then combines with* ***axi*** *to give* ***abx****, so that after assigning* ***a*** *and* ***b****, we cannot assign* ***x****.*
Thus, we have an immediate witness that makes ***x*** *unassignable once we have assigned* ***a*** *and* ***b****.*

Case 5.3.15. *: Let* $\boldsymbol{ax\text{-}z}$*,* $\boldsymbol{bxi}$*,* $\boldsymbol{az\text{-}y}$ *and* $\boldsymbol{ay\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{ax\text{-}z}$ *and* $\boldsymbol{az\text{-}y}$ *into* $\boldsymbol{ax\text{-}y}$ *which would then combine with* $\boldsymbol{by\text{-}i}$ *to give* $\boldsymbol{ax\text{-}i}$ *which then combines with* $\boldsymbol{bxi}$ *to give* $\boldsymbol{abx}$*, so that after assigning* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*.*

Case 5.3.16. *: Let* $\boldsymbol{bx\text{-}z}$*,* $\boldsymbol{axi}$*,* $\boldsymbol{bz\text{-}y}$ *and* $\boldsymbol{ay\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{bx\text{-}z}$ *and* $\boldsymbol{az\text{-}y}$ *into* $\boldsymbol{abx\text{-}y}$ *which would then combine with* $\boldsymbol{by\text{-}i}$ *to give* $\boldsymbol{abx\text{-}i}$ *which then combines with* $\boldsymbol{axi}$ *to give* $\boldsymbol{abx}$*, so that after assigning* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*.*

Case 5.3.17. *: Let* $\boldsymbol{ax\text{-}z}$*,* $\boldsymbol{bxi}$*,* $\boldsymbol{bz\text{-}y}$ *and* $\boldsymbol{ay\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{ax\text{-}z}$ *and* $\boldsymbol{bz\text{-}y}$ *into* $\boldsymbol{abx\text{-}y}$ *which would then combine with* $\boldsymbol{ay\text{-}i}$ *to give* $\boldsymbol{abx\text{-}i}$ *which then combines with* $\boldsymbol{bxi}$ *to give* $\boldsymbol{abx}$*, so that after assigning* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an immediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$ *and* $\boldsymbol{b}$*.*

Case 5.3.18. *: Let* $\boldsymbol{ax\text{-}z}$*,* $\boldsymbol{axi}$*,* $\boldsymbol{bz\text{-}y}$ *and* $\boldsymbol{cy\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{ax\text{-}z}$ *and* $\boldsymbol{bz\text{-}y}$ *into* $\boldsymbol{abx\text{-}y}$ *which would then combine with* $\boldsymbol{cy\text{-}i}$ *to give* $\boldsymbol{abcx\text{-}i}$ *which then combines with* $\boldsymbol{axi}$ *to give* $\boldsymbol{abcx}$*, so that after assigning* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an intermediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*.*

Case 5.3.19. *: Let* $\boldsymbol{ax\text{-}z}$*,* $\boldsymbol{bxi}$*,* $\boldsymbol{az\text{-}y}$ *and* $\boldsymbol{cy\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{ax\text{-}z}$ *and* $\boldsymbol{az\text{-}y}$ *into* $\boldsymbol{ax\text{-}y}$ *which would then combine with* $\boldsymbol{cy\text{-}i}$ *to give* $\boldsymbol{abcx\text{-}i}$ *which then combines with* $\boldsymbol{bxi}$ *to give* $\boldsymbol{abcx}$*, so that after assigning* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an intermediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*.*

Case 5.3.20. *: Let* $\boldsymbol{ax\text{-}z}$*,* $\boldsymbol{bxi}$*,* $\boldsymbol{cz\text{-}y}$ *and* $\boldsymbol{ay\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{ax\text{-}z}$ *and* $\boldsymbol{cz\text{-}y}$ *into* $\boldsymbol{acx\text{-}y}$ *which would then combine with* $\boldsymbol{ay\text{-}i}$ *to give* $\boldsymbol{acx\text{-}i}$ *which then combines with* $\boldsymbol{bxi}$ *to give* $\boldsymbol{abcx}$*, so that after assigning* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an intermediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*.*

Case 5.3.21. *: Let* $\boldsymbol{bx\text{-}z}$*,* $\boldsymbol{axi}$*,* $\boldsymbol{az\text{-}y}$ *and* $\boldsymbol{cy\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{bx\text{-}z}$ *and* $\boldsymbol{az\text{-}y}$ *into* $\boldsymbol{abx\text{-}y}$ *which would then combine with* $\boldsymbol{cy\text{-}i}$ *to give* $\boldsymbol{abcx\text{-}i}$ *which then combines with* $\boldsymbol{axi}$ *to give* $\boldsymbol{abcx}$*, so that after assigning* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an intermediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*.*

Case 5.3.22. *: Let* $\boldsymbol{bx\text{-}z}$*,* $\boldsymbol{axi}$*,* $\boldsymbol{cz\text{-}y}$ *and* $\boldsymbol{ay\text{-}i}$ *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine $\boldsymbol{bx\text{-}z}$ *and* $\boldsymbol{cz\text{-}y}$ *into* $\boldsymbol{bcx\text{-}y}$ *which would then combine with* $\boldsymbol{ay\text{-}i}$ *to give* $\boldsymbol{abcx\text{-}i}$ *which then combines with* $\boldsymbol{axi}$ *to give* $\boldsymbol{abcx}$*, so that after assigning* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*, we cannot assign* $\boldsymbol{x}$*.*
Thus, we have an intermediate witness that makes $\boldsymbol{x}$ *unassignable once we have assigned* $\boldsymbol{a}$*,* $\boldsymbol{b}$ *and* $\boldsymbol{c}$*.*

Case 5.3.23. *: Let* ***bx-z****,* ***cxi****,* ***az-y*** *and* ***ay-i*** *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of ***a****,* ***b*** *and* ***c*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***bx-z*** *and* ***az-y*** *into* ***abx-y*** *which would then combine with* ***ay-i*** *to give* ***abx-i*** *which then combines with* ***cxi*** *to give* ***abcx****, so that after assigning* ***a****,* ***b*** *and* ***c****, we cannot assign* ***x****.*
Thus, we have an intermediate witness that makes ***x*** *unassignable once we have assigned* ***a****,* ***b*** *and* ***c****.*

Case 5.3.24. *: Let* ***ax-z****,* ***bxi****,* ***cz-y*** *and* ***dy-i*** *be four unassignable statements in the differential form closure of the solution space of* F*.*
Then the assignment of ***a****,* ***b****,* ***c*** *and* ***d*** *will not lead to* Blocking Scenario 3 *by the following reasoning:*
Witness correlation will combine ***ax-z*** *and* ***cz-y*** *into* ***acx-y*** *which would then combine with* ***dy-i*** *to give* ***acdx-i*** *which then combines with* ***bxi*** *to give* ***abcdx****, so that after assigning* ***a****,* ***b****,* ***c*** *and* ***d****, we cannot assign* ***x****.*
Thus, we have an intermediate witness that makes ***x*** *unassignable once we have assigned* ***a****,* ***b****,* ***c*** *and* ***d****.*

Remark 5.4. *In this case, we introduced depth-*5 *witnesses, namely* ***acdx-i*** *and the final witness in the sequence,* ***abcdx****. As stated earlier, this is a fill-in-the-gaps inductive theorem proving procedure, and as per our reverse mathematical strategy, we assume that our differential form closure* must *now include unassignable statements for depth-*5 indirect witnesses*.*
We will formalize this assumption into a fact, when we define integration for a system of n *variables in Section 6, in case we discover that we need even higher-depth witnesses to compute the closure. We will also carry this assumption implicitly into all cases that follow after the current one.*

□

Remark 5.5. *The list of cases proven above entails the maximal context of any set of assignments spanning* 4 *different clauses which the theorem addresses.*

5.4 Witnesses Of Mixed Depth

By witnesses of mixed depth, we mean the scenario where one or more of the statements used in the blocking scenarios preexist as *depth-*2 witnesses in the differential form closure ϕ while one or more other statement(s) used in the blocking scenarios preexist as *depth-*3 witnesses in ϕ.
One can easily verify that such a scenario would simplify into one of the following 2 cases:

1. The *depth-*2 witnesses are absorbed into a non-immediate higher-depth witness in the course of computing the differential form closure since they share at least one conflicting assignment with one of the other witnesses. Then the differentiation procedure will not be applied to such a clause.
2. The *depth-*2 witnesses are absorbed into an immediate witness of depth 3 and lower in the course of computing the differential form closure„ since they share at least one conflicting assignment with one of the other witnesses. Then the resulting clause would fall into at least one of the cases treated in the preceding sub-sections.

5.5 Witness Boundary

We have noted from the preceding sub-sections that we will need to include unassignable statements for *depth-*5 witnesses in our closure. We now proceed to the full determination of the witness boundary.

Theorem 5.1. *The differential form closure needs to include unassignable statements for depth-*6 *witnesses.*

Proof. Assume that we have a witness with an unassignable statement **abcdx**, then:

- if ∃ unassignable statements **abcd-x-i** and **abcd-xi**, then by their combination we get **abcd-x** which then combines with **abcdx** to give **abcd**.

- And so, **abcdx** is no longer valid and cannot make true witness statements.

Therefore, the differential form closure needs to include unassignable statements for *depth*-6 witnesses. □

Theorem 5.2. *The differential form closure ϕ does not need to include unassignable statements for witnesses of depth > 6 in order to detect* Blocking Scenarios 2 through 4.

Proof. We never use witnesses of *depth*-6 to produce witness statements for detecting *Blocking Scenarios 2 through 4* therefore, we do not need to validate their implications at *depth*-7. □

We establish the following fact:
Let $C_i, |C_i| > 3$ be a clause in the differential form closure ϕ of $F(\mathrm{n})$. C_i always corresponds to some unassignable statement U_i, and so if S_i is a witness statement deduced from U_i, and $S_i \implies \mathbf{u}$ is a truth assignment to some variable v, then the conjunction ψ, of $\mathbf{u}$ with a set of assignments ρ, **always** occurs in the unassignable statement U_j corresponding to some clause $C_j, |C_j| \leq 3$ because:

- C_i must be ultimately derived from such $C_j, i \neq j$, either directly, OR:
- Via some other clause $C_k, k \neq i, k \neq j$, which itself is ultimately derived from C_j.
- Where C_j OR C_k is integrated with another clause $C_l, l \neq i, l \neq j, l \neq k$ to produce C_i.

Lemma 5.8. *If $\mathcal{A}$ has not encountered **any** blocking scenario at any point in time $t_h, h < i \leq n$, then at time $t_i, t_h < t_i \leq t_n$, then rewriting a clause C_x, $|C_x| \leq 3$, if it is not yet satisfied, will not cause C_x to contradict any assignment that $\mathcal{A}$ has selected.*

Proof. Let the rewrite of C_x force an assignment β to some variable v (Differentiation Rule. 3.3). Since witnesses always correlate, then:

- For any clause C_l: $C_j, C_k,\ |C_j| \leq 3, |C_k| \leq 3,\ C_l = \mathrm{Correlate}(C_j, C_k) \implies |C_l| \leq 4$.
- At time $t = 1$, rewriting C_x ($|C_x|$ has to be 2), if it is not yet satisfied, will not contradict any assignment that $\mathcal{A}$ has selected because $\nexists C_y,\ |C_y| = 1,\ C_y \implies \neg\beta$ at time $t = 1$.
- If $\mathcal{A}$ has not encountered **any** blocking scenario at any point in time $t_h, h < i \leq n$, then at time $t_i, t_h < t_i \leq t_n$:
  - Let $\gamma,\ |\gamma| \leq 3$ be a sequence of assignments that $\mathcal{A}$ has made at any point in time $< t_i$, then:
  - $\nexists C_y,\ |C_y| \leq 4,\ C_y \coloneqq (\gamma \wedge \beta)$ at time $t = 0$.

Therefore, Lemma. 5.8 statement holds. □

The following theorem builds on this lemma and ensures the *global* behavior of $\mathcal{A}$ at all points in time for every possible (explicit and implicit) rewrites. Implicitly rewriting a clause $C, |C| > 3$ means checking Scenarios *Blocking Scenarios 2 through 4* and applying a necessary witness statement from C.

Theorem 5.3. *If the witness boundary has depth*-6 *and $\mathcal{A}$ does not encounter* Blocking Scenario 1 *at time $t = 0$ of differentiation* THEN *$\mathcal{A}$ will not encounter* Blocking Scenario 1 *at any other point in time $t > 0$.*

Proof. We have the following guaranteed behaviors of the proof system:

1. If $\mathcal{A}$ has not encountered **any** blocking scenario at any point in time $t_h, h < i \leq n$, then at time $t_i, t_h < t_i \leq t_n$:
   - If $C_j, |C_j| > 3$ and C_j has unassignable statement U_j, and:
   - If S_j is a witness statement deduced from U_j at time t_i, and $S_j \implies \mathbf{u}$, where $\mathbf{u}$ is a truth assignment to some variable v.

- Explicitly rewriting any (number of) clause(s) $C_k,\ v \in C_k,\ |C_k| \leq 3$ where C_k is not yet satisfied will not contradict any assignment that $\mathcal{A}$ has selected - Lemma. 5.8.
- Therefore, at time t_i, implicitly rewriting C_j will not result in *Blocking Scenario 1* because C_j derives from such at least one such C_k.

2. (Lemma. 5.8 $\wedge$ Item. 1) $\coloneqq \top$ at **all** times.
   Therefore $\forall C, |C| \leq 3$, *Blocking Scenario 1* is always prevented for any variable x at any time by:
   - The application of Differentiation Rules. 3.1 - 3.3 to all applicable clauses at **all** times, and:
   - Detection and prevention of *Blocking Scenarios 2 through 4* at **all** times.

Therefore, $\mathcal{A}$ will not encounter *Blocking Scenario 1* **at any time** $t > 0$ if it does not encounter it at time $t = 0$ of differentiation. [17] □

6 Induced Definitions

We now define the operators for a system of n variables, F(n), based on the proven induction (sub)theorems. For all rules and definitions presented, let $\mathcal{A}$ be the proof system that implements the proof calculus.

First, we add the following rules and definitions for integration for F(n):

Rule 6.1. *In a system of n variables, F(n), that we wish to satisfy, let the following hold:*

- *Let $\mathcal{DFC}$ be the set of all unassignable statements computed by $\mathcal{A}$, so far, by the execution of Rule. 3.4 — pairwise integration of* clauses as schemes.

  Remark 6.1. *The completion of $\mathcal{DFC}$, as a set, is intended to be the differential form closure of F(n), ϕ.*

- *Let U_x and U_y be two distinct unassignable statements in $\mathcal{DFC}$. Let C_x be the clause whose constraint function f_x records U_x and C_y be the clause whose constraint function f_y records U_y. And let Integrated(C_x, C_y) be a predicate that returns "True" if we have executed Rule. 3.4 (pairwise integration) for the pair, C_x and C_y, and "False" otherwise.*

- *Let U_1 be the unassignable statement recorded to a constraint function f_1 for C_1 in F(n), and U_2 be the unassignable statement recorded to a constraint function f_2 for C_2 in F(n), such that Integrated(C_1, C_2) returns "False". Then:*
  - *Execute Rule. 3.4 for C_1 and C_2 and record the result as C_3*
  - *If $|C_3| \leq 6$, add the corresponding unassignable statement, $U_3, U_3 \notin \mathcal{DFC}$ to $\mathcal{DFC}$ — this records U_3 to a constraint function f_3 for C_3.*
  - *Otherwise, discard U_3.*

Definition 6.1. Time t Of Integration*: We assume that $\mathcal{A}$ is a single-threaded algorithm and can only perform one step at a time (in the usual computational sense), which includes the integration of 2 clauses by the execution of Rule. 3.4. Any single execution of Rule. 3.4 is **defined** as occurring at a unique time t of integration. After the execution of Rule. 3.4 is complete at time t then $t + 1$ is the next time of integration, regardless of whether there are any clauses to be integrated or not at time $t + 1$.*

We will delineate times of integration as defined above so that we can continue to use the normal definition(s) of time for differentiation.

[17] This behavior is important to note, because even though the guarantee at the clause level is *piecewise linear*, the collective guarantee is predictable, an argument that will become important later in Section. 9.2.7 when we consider how collections of functions defined using such guarantees ensures a certain type of verifiable computation.

Rule 6.2. *In a system of n variables F(n) that we wish to satisfy, let the following hold:*

- *Let $\mathcal{DFC}$ be the set of all unassignable statements computed by the proof system so far.*
- *At some time t_i of integration:*
  - ***IF*** *$\nexists C_x, C_y$ s.t Integrated$(C_x, C_y) = False$ for any C_x whose constraint function f_x records some $U_x \in \mathcal{DFC}$ and C_y whose constraint function f_y records some $U_y \in \mathcal{DFC}$, then $\phi = \mathcal{DFC}$. In other words, when Rule. 6.1 has been executed for all pairs of statements in $\mathcal{DFC}$, then $\phi = \mathcal{DFC}$, where ϕ is the differential closure of F(n). Label t_i as time t_f of integration.*
  - ***ELSE IF*** *$\exists C_x, C_y$ s.t Integrated$(C_x, C_y) = False$, Execute Rule. 6.2 for pairs C_x and C_y.*
- *At some time t_{null} of integration, if $\exists U_1, U_2 \in \mathcal{DFC}$ s.t $(|U_1| = |U_2| = 1) \wedge (U_1 = \neg U_2)$, then $\phi = \emptyset$. This implies that F(n) is not satisfiable.*

At either time t_f OR t_{null} of integration, the execution of this rule is complete for F(n).

Remark 6.2. *In an efficient algorithm, we can always* spatially *partition the unassignable statements so that if C_x and C_y have zero (no) OR more than one, conflicting truth assignments, then we can consider that Integrated$(C_x, C_y) = True$ and we do not need to actually compare them at a time step.*

With this, we define *integration* for F(n) as:

Definition 6.2. Integration (system of n variables) *: In a system of n variables F(n) and for a DetAlg function $f(t_i)$ at a point in time t_i (as normally defined for the differentiation function), an* Integration *is defined as a function Int$(f(t_i))$:*

- *Which is a morphism $\mathcal{M}$ of an integrable family of schemes $\mathcal{S}$, where:*
  - *$\mathcal{M}$ is embedded within the pullback τ of a Transition Gadget G, $\tau : G_i \times_G G_i \longrightarrow G_i$, $\mathcal{M} \hookrightarrow \tau$.*
  - *The pullback of τ at t_i is:*
    - *The initialization at time $t = 0$ of a set of unassignable statements $\mathcal{DFC}$ with unassignable statements $\{U_n\}$ recorded to constraint functions $\{R_n\}$ for all clauses $\{C_n\}$ in F(n,)* ***AND***
    - *The execution of Rule. 6.2 for $\mathcal{DFC}$ so that the differential form closure ϕ is computed,* ***AND***
    - *If t_i is not time $t = 0$, the completion of the computation of Int$(f(t_{i-1})$.*
- *Int$(f(t_i))$ satisfies Definition. 3.54 (for a small system).*
- *If t_i is not time $t = 0$, then:*
  - *Int$(f(t_i)) = f(t_{i-1})$.*
  - *At all points in time t_j, $0 \leq j < i$, Int$(f(t_j))$ satisfies all conditions listed before this one, recursively.*

    Remark 6.3. *This ensures that ϕ is computed just once and is the same set for all time-indexed differentiations of any (and by implication, all) DetAlg function(s).*

Next, we add the following differentiation rule:
Note: For F(n), we must directly define differentiation w.r.t second-ordered time differentials.

Rule 6.3. *In a system of n variables F(n), with differential form closure ϕ (computed by Integration, as in Definition. 6.2, at time $t = 0$ Of differentiation), that we wish to satisfy, let the following hold, at time t, with a DetAlg $\mathcal{D}$:*

- *$\mathcal{A}$ has selected a truth assignment ψ for some variable v at time t.*

Then do the following:

- *Add ψ to $\mathcal{D}$.*
- *For any clause $C, v \in C.\neg(C := \top), |C| = k, k \leq 3$, rewrite the unassignable statetemt U recorded to the constraint function of C according to Rules. 3.1 - 3.3 s.t $|C|, |U| = k - 1$.*
- *If $\exists U$, an unssignable statement, where $|U| = 1$, assign $\neg U$ (Rule. 3.3), advance t to $t = t+1$ and repeat this rule - Rule. 6.3 (for the first of such U found, to properly delineate block of time incrementally).*
- *Check ϕ for any assignment $\boldsymbol{a}$ that will cause Blocking Scenarios 2,3 or 4.*
- *For the first such a found, advance t to $t = t + 1$ and select $\neg\boldsymbol{a}$ and repeat this rule - Rule. 6.3 (block of time delineated incrementally).*

And with this, we define differentiation for F(n):

Definition 6.3. Differentiation (system of n variables) *: In a system of n variables $F(n)$ with differential form closure ϕ and for a DetAlg function $f(t_i)$ at time t_i, a* Differentiation *is defined as a function Diff$(f(t_i))$ for which the following holds:*

- *ϕ is computed by Int$(f(t_0))$.*
- *Diff$(f(t_i))$ is a morphism $\mathcal{M}$ of a differentiable family of schemes $\mathcal{S}$, where:*
  - *$\mathcal{M}$ is embedded within the pushforward τ of a Transition Gadget G, $\tau : G_i \times_G G_i \longrightarrow G_i$, $\mathcal{M} \hookrightarrow \tau$.*
  - *The pushforward of τ at t_i is:*
    - *The assignment $v := \psi$ for some variable $v, \psi = \{0 \oplus 1\}$, **AND***
    - *The execution of Rule. 6.3 for F(n) at t_i.*
- *Diff$(f(t_i))$ satisfies Definition. 3.46 (for a small system) for any clause C where $v \in C$ **AND** $U \in \phi$, is the unassignable statement recorded to a constraint function R for C **AND** $|U| \leq 3$.*
- *If t_i is not time $t = n$, then Diff$(f(t_i)) = f(t_{i+1})$.*
- *If t_i is not time $t = 0$, then at all points in time t_j, $0 \leq j < i$, Diff$(f(t_j))$ satisfies all the conditions listed before this one, recursively.*

Note that both the differential form closure ϕ and the two operators are *twisted* by definition.
The reader can verify that these definitions satisfy all the properties we have noted for both operators, in the small limit of the solution space of a single clause — since they simply extend/subsume the definitions of the (small system) operations that were used to obtain these properties, via a topological ultraproduct construction. Particularly important is that each operator is defined with respect to its categorical limits (a second order *meta-model*) via the topological closure of the ultratopology which corresponds to the differential form closure. In the next section, we will verify that the differential form closure itself is logically sound and complete and that it accurately dualizes both operators at all times.

7 Fundamental Theorems

We now establish the calculus as sound and complete via 2 fundamental *second-order* theorems. For convenience in following the logic, we also provide a pseudo-code implementation of the calculus as a proof system in a separate subsection following the subsections on the theorems.

7.1 First Fundamental Theorem

Theorem 7.1. *For a system of n variables, F(n), for any function f which computes a set of satisfying truth assignments to the clauses in F(n), Int(f), where Int(f) is the operator defined by Definition. 6.2, is computable by the proof calculus, and Diff(f), where Diff(f) is the operator defined by Definition. 6.3, is as well computable by the proof calculus.*

Proof. Our proof assumes that the $F(\mathrm{n})$, used in the proof, if satisfiable, has more than one satisfying solution. We will present the case of an $F(\mathrm{n})$ with just one solution in the course of proving the larger instance of $F(\mathrm{n})$.
We present nested supporting lemmas for each distinct statement about soundness or completeness.

Let $\ominus$ be a proof system that implements Definition. 6.3 (large differentiation) and Definition. 6.2 (large integration) for $F(\mathrm{n})$ (such as our pseudo-code listing in Section. 7.3), then the following soundness and completeness arguments hold:

1. **Soundness**

   Lemma 7.1. $\ominus$ *will compute a set of witnesses* $\{\mathcal{W}\}$ *for F(n) which can be used to derive a valid solution to F(n).*

   Proof. As follows:
   Let $\{\mathcal{W}\}$ be the set of witnesses with corresponding unassignable statements $\{\mathcal{U}\}$, computed by $\mathrm{Int}(f_0)$ for $F(\mathrm{n})$ at time $t = 0$ of differentiation.

   (a) If
   $$\nexists U_i, U_j \in \{\mathcal{U}\}, i \neq j, U_i = \neg U_j.$$
   Then using $\{\mathcal{W}\}$, we can derive a *DetAlg* $\mathcal{D}$ through the use of free assignments and witness statements as ensured by the correctness of the inductive subtheorems and lemmas.

   (b) $\mathcal{D}$ must satisfy every clause in $F(\mathrm{n})$ as ensured by the fact that it satisfies Definition. 3.46 (small differentiation).

   (c) Therefore, $\{\mathcal{W}\}$ can be used to derive a valid solution to $F(\mathrm{n})$.

   □

2. **Soundness Corollary**

   Corollary 7.1. $\ominus$ *will not compute a differential form closure for some F(n) which is unsatisfiable.*

   Proof. If at any point in time of integration:
   $$\exists U_i, U_j \in \{\mathcal{U}\}, i \neq j, (|U_i| = |U_j| = 1) \wedge (U_i = \neg U_j).$$
   Then $F(\mathrm{n})$ is not satisfiable. □

3. **Completeness**

   Lemma 7.2. *If DetAlg*$^\omega$ *is an arbitrary satisfying solution to F(n), then it is computable by* $\ominus$*, that is,* $\ominus$ *can compute all satisfying solutions to F(n).*

   Proof. We prove by contradiction:

   (a) Assume $\exists$ *DetAlg*$^\omega$ not computable by $\ominus$.

   (b) Adjust $F(\mathrm{n})$ so that it has only 1 solution:

   i. Add a set X of 3-SAT clauses where for any variable $v \in X \implies v \in F(\mathrm{n})$

ii. For each clause C in X, create 6 other variations of C such that only the assignments in $DetAlg^{\omega}$ can satisfy all 7 clauses.[18] Doing this, we obtain a formula F_n^{ω}.

(c) Now the following must hold for F_n^{ω} for the integration of f_0 at time $t = 0$ of differentiation:

$$\text{Int}(f_0) = DetAlg^{\omega}.$$

Since each variable v must correspond to an auto-differentiated assignment $\psi, \psi \in DetAlg^{\omega}$ — $\ominus$ only makes sound judgements about unassignability at the clause level since it satisfies Definition. 3.54 (small integration). [19]

(d) This means no assignment ψ_i blocks any other assignment $\psi_j, i \neq j, \psi_i, \psi_j \in DetAlg^{\omega}$.

(e) Since $F_n \subset F_n^{\omega}$, then $\exists f$ for F_n s.t Diff$(f) = DetAlg^{\omega}$

(f) Therefore, our starting assumption is false.

□

Therefore, differentiation is sound for any f and integration is complete for all f for F(n).

□

7.2 Second Fundamental Theorem

Theorem 7.2. *For a system of n variables, F(n), for any differentiable assignment function f: $I(D(f)) = f$ and $D(I(f)) = f$ where D is defined by Definition. 6.3 and I is defined by Definition. 6.2.*

Proof. If all variables in F(n) have auto-differentiated assignments, then this duality is implicit. If F(n) has more than one solution, then we can always invert $D(f)$ back to the time point t of every *Free-Val* and thus back to time $t = 0$ of differentiation. Therefore, $I(D(f)) = f$ and $D(I(f)) = f$.[20] □

7.3 Pseudo-Code Implementation Of Proof System

The following algorithm outlines a straightforward, proof system implementation of the calculus. It bundles the two (delineated) operators together into a single execution that accepts a 3-SAT formula, F, as input. The reader can verify that the steps made by the algorithm correctly implement Definitions 6.2 and 6.3.

[18] It is trivial to show that a set of 7 different clause permutations of a 3-SAT clause can only be satisfied by one set of assignments, with each new permutation, the number of satisfying solutions decreases from the number which satisfies a single clause, that is, 7. A set of 8 different permutations is unsatisfiable.

[19] And will not mark an assignable statement S, $|S| = 1$ as unassignable, that is, witness correlation is sound for any pair of clauses.

[20] This dualization over n-forms is equivalent to the Hodge star duality between derivatives and exterior derivatives. This, along with the underlying (hyper)Kähler form and the presence of codifferentials suggests a strong link with algebraic Hodge theory, such that problems in that theory can be mapped back to this one via a type of *reverse* ultraproduct construction, and studied in a more discrete setting.

Input: A 3-SAT formula F over variables $\{x_1, \ldots, x_n\}$ with clauses $C_1, \ldots, C_m$.
Output: Either a *DetAlg* with satisfying assignment, or "UNSAT".

Initialization:
$D \leftarrow \emptyset$; `// Queue of unprocessed witnesses (unassignable statements)`
$S \leftarrow \emptyset$; `// Set of processed witnesses (unassignable statements)`
$A \leftarrow \emptyset$; `// DetAlg under construction`
$B \leftarrow 6$; `// Witness Boundary Depth`
RandomAssignment$(v_x) \rightarrow$ RandomChoice$(\{x, \neg x\})$; `// Utility function for random assignment to a variable` v `with subscript` x.
for $i \in \{1, \ldots, m\}$ **do**
  $U \leftarrow$ ClauseToWitness(C_i) ; `// Convert clause` C_i `into initial unassignable statement`
  insert U into D
end

Phase I: Integration
while $D \neq \emptyset$ **do**
  remove d from D;
  foreach $s \in S$ **do**
    if *d and s share exactly one conflicting literal ℓ* **then**
      $r \leftarrow$ Correlate(d, s) ; `// Combine and cancel conflict` ℓ
      if $|r| \leq B$ *and* $r \notin S \cup D$ **then**
        insert r into D;
      end
    end
  end
  insert d into S;
  if $|d| = 1$ **then**
    if $\exists \neg d \in A$ **then**
      return *"UNSAT"* ; `// Empty differential form closure` $\implies$ `contradiction found`
    end
    else
      insert r into A; ; `// Auto-differentiate` d
    end
  end
end

Phase II: Differentiation
while $|A| < n$ **do**
  if $\exists f \in S, |f| = 1$ **then**
    $a \leftarrow f$;
  end
  else
    choose an unassigned variable v_x;
    $a \leftarrow$ RandomAssignment(v_x);
  end
  insert a into A;
  foreach $d \in S$ **do**
    $d \leftarrow$ Reduce(d, a) ; `// Apply assignment` a `to rewrite` d
  end
  foreach *ℓ which is a truth assignment to an unassigned variable $v_y, x \neq y$* **do**
    if CheckBlocking(S, ℓ) **then**
      insert $\neg \ell$ into S;
    end
  end
end

return *DetAlg* A;

Algorithm 1: Proof System

Subroutine definitions are included in the Appendix. B.

8 Computational Complexity Of The Calculus

8.1 Runtime Analysis

We reference the (straightforward) implementation of the calculus within a proof system, as presented in Algorithm. 1, to analyze the worst-case runtime scenarios of the operations of the calculus.

Definitions.

- F: 3-SAT formula to be satisfied.
- n: number of propositional variables in F.
- m: number of clauses in F. We will not need this parameter since input size for worst case analysis can be performed using just n as a combinatorial parameter (demonstrated below).
- B: *witness boundary*, defined as the maximum cardinality of any unassignable statement (witness) generated during Integration. For our analysis $B = 6$ as proved.
- N: number of distinct witnesses generated during Integration, after duplicate-elimination.
- *Correlate*(x, y): witness correlation as defined by the Algorithm. 1.
- Ψ: total number of comparisons used for witness correlation.

8.1.1 Integration

The two prominent factors that determine the upper bound behavior of the integration fragment are:

- *Correlate*(x, y) is performed over every pair of witnesses[21] and so $\Psi = N \times N$ comparisons.
- Since the depth of any witness $|W| \leq B$:

$$N \;\leq\; \sum_{k=1}^{B} \binom{2n}{k} \approx C \times n^6, \text{where } C \text{ is some constant.}$$

The maximal running time for integration can be computed in a straightforward way, but varies for different actual implementations, depending on the following particular implementation detail - how duplicate witness unassignable statements generated during the integration of clauses are detected:

1. If the implementation stores unassignable statements with a hash structure, then looking up a duplicate unassignable statement will happen in constant time. We can complete all needed comparisons in $\boldsymbol{N} \times \boldsymbol{N}$ steps, that is, $\left(C \times n^6\right) \times \left(C \times n^6\right) \approx O(n^{12})$. We will also allow for unassignable statements generated more than once (which is why we check for duplicates). Let E represent the average number of times any unassignable statement can be generated, within some Gaussian distribution. And so: $\Psi \approx E \times C^2 \times n^{12}$. Factoring out the constants, we get a worst-case runtime of $O(n^{12})$.
2. If an efficient sorting procedure is used to store unassignable statements, that is, duplicates can be looked up in *log* n time, then each new unassignable statement will need to be checked against already generated unassignable statements in (worst case) *log* n^6 times which approximates to adding, on the average, an extra number of n steps for comparison checks for each new unassignable statement. And so, we perform $\approx C \times n^7$ checks (with random constant C) for each comparison of a new unassignable

[21] As noted in Section. 6, we can always partition the space of witnesses effectively so that only applicable comparisons are actually performed.

statement with elements of size $\approx C \times n^6$ and so $\Psi \approx E \times C^2 \times n^7 \times n^6$. Factoring out constants, we obtain a worst-case runtime of, following the same analysis pattern as above, $O(n^{13})$.

3. Assuming no sorting procedure is used to store unassignable statements, that is, duplicates can only be looked up sequentially, then each new unassignable statement will need to be checked against already generated unassignable statements in (worst case) n^6 times which approximates to adding, on the average, $C \times n^6 \times n^6$ steps for each comparison of a new unassignable statement with elements of size $\approx C \times n^6$. Following a similar analysis as above, this gives $\Psi \approx E \times C^2 \times n^6 \times n^6 \times n^6$ incurring a worst-case runtime of, with constants factored out, following the same analysis pattern as above, $O(n^{18})$.

8.1.2 Differentiation

Differentiation can also be analyzed w.r.t if the differential closure/set of unassignable statements, ϕ, is sorted (with one of the sorting criteria being witness depth/clause size) or not:

- If ϕ is sorted, then differentiation assigns truth values to n number of variables in linear time resulting in a $O(n)$ worst-case runtime. Checking for blocked variables requires going through at most $\approx n^2$ clauses at most n (decreasing by 1 at each time step) times to give a worst-case runtime of $O(n^3)$. Rewriting $\binom{n}{3}$ clauses in the differential core at most n number of times (assignments) incurs a worst-case runtime of $O(n^4)$. Therefore, the worst-case runtime for differentiation in this case is $O(n^4)$.
- If ϕ is unsorted, then all 3 distinct subprocesses identified above require going through every clause in ϕ where $|\phi| = B$ in linear time (in $\approx n^6$ steps) for every assignment, totaling n, resulting in a worst-case runtime of $O(n \times n^6)$, and so, the overall worst-case runtime for all operations is $O(n^7)$.

8.2 P VS NP

If complexity is regarded simply as spacetime complexity, typically referred to in shortened form, as time complexity, then the runtime analysis presented above shows that both differentiation and integration are implementable in polynomial time. This settles the question of *P VS NP* in a positive light, namely that *P = NP*.

There is another view that one can take - which is that *NP* problems require a vastly more expressive logic to underlie their algorithms, that is, they are unequal in terms of descriptive complexity. One may be inclined to then conclude that this implies that the two problem classes require vastly different computational models. We prove this to be true in the next section where we explore the computing model which underlies our meta-algorithm..

So how does one interpret this result? We maintain that the answer to that question solely rests on the semantics attached to the equality operator in the phrasing of the conjecture and that although it appears that we have solved the question at one level, that is, the algorithmic, deeper arithmetic questions may still remain which may be resolvable by explicitly designating the different procedures to an arithmetic hierarchy via some applicable arithmetization scheme. We hope to address this point in a future work.

9 Implications For Computability Theory

We now briefly examine the notion of computability as it is normally found in the mathematics and computer science literature as implied by the Church-Turing Thesis and the consequences that our Proof calculus implies for it.

9.1 Church-Turing Thesis In Set Theoretic Terms

Rather than examining the Church-Turing thesis in full detail along with its technical specifications and history, we will investigate it chiefly by its implications within set theory. We assume that the reader is

familiar with workings of a Turing machine.

First, we state what the thesis implies about function computability:

- A Turing machine T is an automaton capable of enumerating some arbitrary subset S of valid strings of an alphabet A.
- These enumerable subsets $\{S\}$ correspond to the programs that can be executed on T.
- A T enumerated program P can be encoded as a finite string over a finite alphabet (states correlated to symbols).
- Which makes the set of T enumerable programs, $\{P\}$, a subset of the set of finite strings over a finite alphabet $\{S_A\}$.
- $\{S_A\}$ is countable.
- Therefore, $\{P\}$ is countable.
- By the above, the set of all Turing machines $\{T\}$ is countable by a mapping: $\{P\} \to T$.
- The set of computable functions is the set of functions computable by a Turing Machine (Church-Turing thesis).
- Each P computes at most one function.
- Therefore, the set of computable functions Γ is the image of a countable set $\{T\}$ under a one-to-one mapping so that:
$$|\Gamma| = |\{T\}| \times |\{P\}|.$$
— And so, the set of computable functions is countable, that is, has cardinality $\aleph_0$ (Set-Theoretic Implication of the thesis).

Next, we specify a set-theoretic universe, Λ-universe, which consists of all possible λ-functions, by construction:

Definition 9.1. Λ-universe *: We define a* Λ-universe *by the following construction:*

1. *Let* Λ-universe *contain infinitely many formulas, whose cardinality is as of yet unspecified.*
2. *Let* $\mathcal{F}$ *be the set of all satisfiable formulas in* Λ-universe*.*
3. *Let all variables exists in a* Λ-universe*, that is,* $\exists.v_i, i \in \mathbb{N}$*, and so,* $|\{v_i\}|$ *has cardinality* $\aleph_0$*,*
4. *Every combination of truth assignments to any finite subset of* $\{v_i\}$ *satisfies at least one formula,* $\mathcal{F}_k \in \mathcal{F}, k \in \xi$ *where* $|\xi|$ *is to be determined.*

Theorem 9.1. *The cardinality of both the set of inputs* α *(sets of possible* λ*-functions) and set of outputs* β *(a single* λ*-function) to all 0-form* λ*-functions in the* Λ-universe *is* $2^{\aleph_0}$*.*

Proof. Both the set of inputs and set of outputs of all *0*-form λ-function, computable by the proof calculus, are in one-to-one correspondence with the powerset of the set of all variables, $\{v_i\}$, in Λ-*universe*, simply because there must be at least one formula that is satisfied by any combination of truth values to any finite subset of $\{v_i\}$, as stated above in the definition of the Λ-*universe*. A ***computation*** is then definable as an arbitrary sequence (up to infinite size) of such finite subsets. Such computation happens at every point in time along a trajectory - a (set of) lambda function(s) is computed as the output of the current step and the input (if applicable) of the next computation [22]. By this definition, the total cardinality of computable functions is of size $\prod_{i=0}^{\infty} F_i$, where F_i is some finite subset of assignments. In this case, a λ-function is redefined as any element of this product (is a any valid sequence of finite λ-functions.
This fixes the cardinality of both the set of inputs and set of outputs*0*-form λ-functions as $2^{\aleph_0}$. □

[22] So that for our model, computation is a path and not a destination as in a standard Turing machine

Corollary 9.1. *A 0-form λ-function is a continuous real-valued function:* $\mathbb{R} \to \mathbb{R}$.

Proof. Each element of the 2 sets – inputs α and outputs β of a *0*-form λ-function, corresponds to a real number – $\alpha_i \in \alpha \equiv \alpha_i \in \mathbb{R}$ and $\beta_i \in \beta \equiv \beta_i \in \mathbb{R}$ — since $|\alpha| = |\beta| = |\mathbb{R}| = 2^{\aleph_0}$. □

We prove that all intermediate λ-functions are also real-valued in A.2.

Theorem 9.2. *The cardinality of the set of all computable λ-functions* $\{\lambda_c\}$ *in the* Λ-universe *is* $2^{\aleph_0}$.

Proof. A *0*-form λ-function $\lambda_c \in \{\lambda_c\}$ continuously maps a finite subset of possible *0*-form λ-functions Λ^F, as inputs, to one *0*-form λ-function, that is, itself, as output. This mapping is reversible. Because all possible subsets of Λ^F are used as input by at least one λ_c-function, there is a bijection between Λ^F and $\{\lambda_c\}$. By the Schröder–Bernstein theorem, these 2 sets have the same cardinality, which is $2^{\aleph_0}$. Therefore, the cardinality of $\{\lambda_c\}$ in the Λ-*universe* is $2^{\aleph_0}$. □

Remark 9.1. $|xi|$ *is* $2^{\aleph_0}$ *since more than one formula, that is, m formulas, can be satisfied by any combination of up to infinitely many truth assignments (we are allowing now for infitely sized formulas). m however, can only be a polynomial multiple of n, the number of variables in the* 3*-SAT formula, since* $\binom{n}{3}$ *is polynomial. There must also exist at least* $2^{\aleph_0}$ *number of formulas that are satisfied by only one λ-function. From these observations, we can conclude that* $|xi|$ *must be a polynomial-valued multiple of* $2^{\aleph_0}$. *We prove in A.1 that* $\mathcal{F}$, *the set of all satisfiable* 3*-SAT formulas, corresponds to* $\mathbb{C}$, *also of cardinality* $2^{\aleph_0}$.

Theorem 9.3. *The proof calculus computes some functions that are not computable as per the Church-Turing thesis.*

Proof. As per the Church-Turing thesis, the cardinality of the set of all computable functions is $\aleph_0$.
But the set of all *0*-form λ-functions computable (given by a deterministic algorithm) by our Proof Calculus in the Λ-*universe* is $2^{\aleph_0}$ (Theorem. 9.2). □

This implies that $\exists f$ at least one function (like the one that encodes a proof system that implements our calculus, implementable on a machine constructed using a Von Neumann architecture) — of which there may be infinitely many symbols (variable subscripts) and states (integrations)— but finitely many rules of *exact computation* over *types/classes* of symbol combinations — which have no direct mapping to a unique symbol — for example, rules dealing with clauses with at most one conflicting assignment — where f is not computable by a Turing Machine, that is, a machine whose enumerations can be encoded as finite strings over finite alphabets. In other words, we upgrade from the usual definition of a symbol as a bit/bit-string to a symbols as a formula/type which may have an infinite identity (univalence). [23]

We now give a more "concrete" example of a computable function (not a λ-function), that demonstrates the truth of Theorem. 9.3, by addressing the *Halting Problem* from the standpoint of our calculus.

9.2 The Halting Problem

The validity of the claims made about the halting problem in this section are *contingent* on the acceptance of an extra-logical axiom that we derive in Section. 9.2.5, from direct observations of the logical properties of the calculus. As an axiom for our proof calculus, this axiom is actually meta-logical.

In addition, one must allow our framework full second-order quantificational scope over all computable functions in our universe. This should be viewed as similar to the use of an infinite tape in the standard Turing machine. In our case, finite sections of an infinite computation are both independent in space (like a Turing machine) and in time - they can be theoretically computed in parallel(unlike a Turing machine).

The halting problem, stated briefly, is the problem of determining, from a description of an arbitrary computer program P, and an input I, whether P will finish running at some time t, or run forever.

[23] That is, we implicitly implement a type of univalent higher order homotopy type theory

9.2.1 The Argument For Uncomputability

The usual argument for the uncomputability of the halting problem relies on the following key assumptions:

- That computable functions are computed by some Turing enumerable program.
- That computable functions can *invoke* each other.
- That computable functions can take any kind of input, including source code of programs.

With these assumptions, one can establish the argument for the uncomputability of the halting problem as follows:
Let *Tur* be a Turing Machine and the function notation *f(a, b)*, denote a function f computable by some program z in *Tur*, that takes 2 arguments, a and b, where a is a program in *Tur* and b is an input to a where b can be another program in *Tur*, including z.
And let i be an argument for a program which refers to the i^{th} program in an enumeration of all the programs of *Tur*.
And Let x be an arbitrary input.
And let H be the *assumed* (total) computable halting function for program i on input x:

$$H(i,x) = \begin{cases} 1 & \text{if program i halts on input x} \\ 0 & \text{otherwise} \end{cases}$$

And let p be a program that computes the *partial* function g defined below:

$$g(i) = \begin{cases} 0 & \text{if } f\,(i,i) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Now if we pass p, the program that computes g, as an argument to g, the one of the following must hold:

- *f(p, p)* $= 0$ and so *g(p)* $= 0$. Since p halts, *H(p, p)* $= 1$.
- *f(p, p)* $\neq 0$ and so *g(p)* is undefined. Since p does not halt, *H(p, p)* $= 0$.

In either case we show that H cannot be any f computable by *Tur*. The main issue arises because we allow g, access to its own context, the source code of the program that computes it.

9.2.2 The 3-SAT Assignment Forcing Problem

In order to address the halting problem from the standpoint of our calculus we first describe a different kind of problem which we will call the 3-SAT Chiral Semiprime Factoring Problem.

Definition 9.2. 3-SAT Factor Of A Chiral Semiprime - Type I *: We define a* 3-SAT Factor Of A Chiral Semiprime *of a* 3*-SAT problem F as follows:*

- *Let* Γ *be a set of assignments to variables in* F*.*
- *Turn* Γ *into a* 1*-SAT formula: Make each of the assignments a separate clause and create a formula,* G *of conjunctions of these separate clauses.*
- *Create a formula F(G) by a conjunction of* F *and* G*.*
- *Integrate F(G) (via the proof calculus).*

If all variables in F(G) are auto-differentiated, then we say that G*, the* 1*-SAT representation of* Γ*, is a* 3-SAT Factor Of A Chiral Semiprime - Type I *of* F*.*

We say that F is chiral in the sense that G is a prime factor of F, as a semiprime, *in the direction* of the auto-differentiation of the solution space of F by G. We state factors as formulas instead of as sets of assignments so that we can define even more complex type of factors, as in the next definition.

Remark 9.2. *This version of prime factorization is stronger than the usual one used for multiplication circuits in traditional* 3*-SAT solving. There, prime factoring (and factoring in general) of a semiprime, simply means that a set of assignments, as bit-strings (with delineating positions for both (all) factors) will satisfy the multiplication circuit encoding of the output number to be factored. In our case, in an analogy with the normal SAT solver case, we* **require** *that the set of bits of* **only** *one of the inputs completely determines the bits of the other input (of the circuit), which, for our calculus, will always be the case for an actual semiprime factor (all bits of the second number will become bound as per the logic of the calculus).*

We know that the halting problem allows for passing non-trivial inputs, that is, complete programs, to functions, so we will add a more complex type of factor for a chiral semiprime.

Definition 9.3. 3-SAT Factor Of A Chiral Semiprime - Type II *: We define a* 3-SAT Factor Of A Chiral Semiprime *of a* 3*-SAT problem F as follows:*

- *Let E be a* 3*-SAT formula which is not a* $1-SAT$ *formula.*
- *Create a formula F(E) by a conjunction of E and F.*
- *Integrate F(E).*

If all variables in F(E) are auto-differentiated, then we say that E is a 3-SAT Factor Of A Chiral Semiprime - Type II *of F.*

Remark 9.3. ***Note*** *that* 1*-SAT clauses are also automatically* 3*-SAT clauses by definition.*

Theorem 9.4. *An algorithm ordered by a* First Order Temporal Model, *TM-1 cannot solve the* 3-SAT Factor Of A Chiral Semiprime Problem.

Proof. This follows from Theorem. 3.1 that *TM-1* cannot be used to order a *DetAlg* for 3-SAT because it lacks a model of unassignability and cannot determine from a given set of assignments A, what set of assignments B become *bound* in a *DetAlg* that include the assignments in A. □

Given the validity of the statement of Theorem. 9.4, we say that *TM-1* does not express a valid 3-SAT assignment forcing model. The proof calculus does have a valid 3-SAT assignment forcing model.

A 3-SAT assignment forcing model implies that given any partial set of assignments Γ to a 3-SAT formula F, we can determine what set of assignments Ψ need to be forced to keep F satisfiable. We **define** the problem of determining a 3-SAT assignment forcing model, the 3-SAT Assignment Forcing Problem.

Note that the 3-SAT Assignment Forcing Problem is a *sub-problem* of the 3-SAT Factor Of A Chiral Semiprime Problem.

9.2.3 Overview Of Simple Logic Programs

A Horn Clause is a clause with the following implication form:

$$g \longleftarrow (p_1 \wedge p_2 \cdots \wedge p_n)\,.$$

Which is used to prove that a conclusion g, called the goal, is implied by the conjunction of all the promises $p_1....p_n$. The above implication form can be rewritten in disjunctive form as:

$$\neg p_1 \vee \neg p_2 \cdots \vee \neg p_n \vee g.$$

For *elementary* horn clauses, each p_i is an atomic formula of the form $f(a_1, \ldots, a_n)$, which is a predicate that returns true for arguments $a_1, \ldots, a_n$, indicating a relation among its arguments. For example:

$$twoTimes(4, 2).$$

Is a valid atomic formula p of arity 2 stating a fact and:

$$fourTimes(X, Z) \coloneqq twoTimes(X, Y),\ twoTimes(Y, Z).$$

Is a valid representation of a horn clause $(\neg A \vee \neg B \vee g)$ where:

$$A \coloneqq twoTimes(X, Y),\ B \coloneqq twoTimes(Y, Z).$$

And g is an affirmed goal. This evaluates to:

$$\neg(\neg A \vee \neg B) = (A \wedge B) \implies g.$$

Which is a form of resolution, and in our case, Rule. 3.3.

A simple horn clause logic program LP can consist of one or more elementary horn clauses. The inputs to LP are queries which can be expressed as:

$$fourTimes(X, 3) \text{ or } fourTimes(12, Z).$$

Simple horn clause logic programs are Turing Complete [36, 37]. A simple horn clause logic program LP can also be transformed into a k-SAT formula which in turn can be transformed into a 3-SAT formula $\mathcal{X}$. We can also define the notions of total and partial inputs to LP.

Definition 9.4. Total Input *: A total input t to a simple horn clause logic program LP is a set of arguments such that every atomic formula in each horn clause of LP is mapped to a query in t.*

Definition 9.5. Partial Input *: A partial input P to a simple horn clause logic program LP is a set of arguments such that* not *every atomic formula in each horn clause of LP is mapped to a query in P.*

9.2.4 Termination Of Simple Logic Programs - Total Functions

The halting problem for a simple horn clause logic program LP w.r.t a total function simply asks if LP with a set of horn clauses $\{\mathcal{H}_k\}$, will terminate, given total input I, which means that, for each horn clause $\mathcal{H}_k \in \{\mathcal{H}_k\}$:

- $\mathcal{H}_k$ contains n premises.
- The goal fails: At least one of the premises of $\mathcal{H}_k$ *in disjunctive form* is true : $\exists p_i, \neg p_i \coloneqq \top$, $1 \leq i \leq n$, **OR**:
- The goal passes: All of the premises of $\mathcal{H}_k$ are true and so all premises *in disjunctive form* are false: $\neg(\bigvee_{i=1}^{n} \neg p_i \coloneqq \bot) \equiv \bigwedge_{i=1}^{n} p_i \coloneqq \top$, $1 \leq i \leq n$.

This is equivalent to checking if the K-SAT conversion of LP is satisfied by I. If we ignore both self-referential and self-contradictory *function* (program) inputs (which we will address in Section. 9.2.7), then in cases where *either* of these conditions hold, we say that LP halts, and in cases where *neither* of the conditions hold, we say LP does not halt, so that, if we ignore both self-referential and self-contradictory function inputs, then:
Let $HaltTot(X, Y)$ be the *assumed* (total) computable halting function for a (simple logic) program X on total input Y.
$\forall H_k \in \{H_k\}$ for $p_i \in H_k, |H_k| = n$, $1 \leq i \leq n$

$$HaltTot(X,Y) = \begin{cases} 1 & \text{if } (\exists p_i, \neg p_i \coloneqq \top \quad 1 \leq i \leq n) \vee \left(\left(\neg \left(\bigvee_{i=1}^{n} \neg p_i \right) \coloneqq \top \right) \equiv \left(\bigwedge_{i=1}^{n} p_i \coloneqq \top \right) \right) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Note that this is different from the *HORNSAT* problem which asks if a conjunction of (just) horn clauses, without inputs, is satisfiable.

9.2.5 Action Of A Proof System

We briefly discuss the notion of the proof system, not just as a physical symbol system, but as a *logical simulation* of an actual physical system. This notion will help clarify some of the arguments we will use in the following sections. We develop the notion of the action of the proof system.

Lemma 9.1. *A forced assignment is a logical implication by any valid system of logic, L.*

Proof. By the following counter-argument: Assume a forced assignment is not a logical implication. Then:

- Assume F is a formula for which the calculus forces assignments, but the forced assignments are not logical implications.
- Also assume that at time t, the calculus forced a set of assignments β from an input set of assignments α.
- Then there exists some system L that can infer some other forced assignment $a \notin \beta$.
- This means the *DetAlg* computed by the calculus will encounter a blocking scenario if we select $\neg a$, which is allowed by the calculus.
- But we know the calculus is sound and will encounter no such blocking scenario.

Therefore, a forced assignment is a logical implication by any system of logic, L. □

Remark 9.4. *Thus, the calculus extends the first order logic program (logic programs quantify the variables that comprise their atomic formulas) by a second-order.*

Remark 9.5. *With respect to the above lemma, we can conclude that the proof calculus over the clauses of a 3-SAT formula F in conjunction with any particular input I, is a system of interactions within a closed* ***spacetime****, that is, it is a* ***field theory****.*[24] *As such, if an assignment is not forced by the action of the meta-algorithm, it cannot be* ***forced*** *via any other means, either by some logical condition within, or outside of the proof system.*

Action, as used as a term for physical systems, typically refers to a functional attached to a minimum/least point which is a saddle point. It is usually defined as the integral of a *Lagrangian* L for some evolution function E between two points in time $\mathbf{q}(t_1)$ and $\mathbf{q}(t_2)$. L for classical systems is usually denoted as $L = T - V$. Where T is the kinetic energy and V is the potential energy such that their difference at any point in time describe E. In our system, we can take E at any point in time t_i, to be a λ-function computed

[24] A twisted field theory of doubly-fibrated transforms such as the twistor algebra [38] transforms [39, 40] of Penrose, in physics. A complete analogy would place clause-level constraint functions ψ as massive fermions which twist (twistors) the complexified spacetime, which itself corresponds to the trajectory of a λ-function. This analogy would also place the meta-algorithm as the bosonic spinor (bundle), a double cover of orthogonal spacetime projections which travel (setting an upper limit on possible motion) in complexified null (zero-length) spacetime. As such, the ψ-functions exert a *force* of gravity on each other transmitted by bosons with spin. We hope to flesh out this field-theoretic perspective in future work, but we use it here to solidify the fact that *constraint functions, supplemented with or without inputs, in our proof calculus, exert a (non-brute) force on each other.*

by the *(group structured) action* of a transition gadget G_i.[25] The action of any such G_i is achieved at every point in time as a correspondence with the integration I over the differential forms computed by the calculus at t_i. I at t_i is exactly the morphism embedded in the pullback of G_i (along the transverse manifold)s and is the action of the system.

Definition 9.6. *We **define** the action of a proof system as the integration of forms expressed in Eqn. 7, at each point in time.*

We formalize the observations made in the above remarks, and supported by both our definition of an action, as well as by Lemma. 9.1, into an axiom of *exhaustive force*. This axiom is *extra-logical*, as it involves the idea of the proof calculus as a quasi-*physical* system whose *properties* are required to compute the logic of the calculus. Like we stated earlier, in the particular case of our proof calculus, it is meta-logical.

Axiom 9.1. *The only element which can force an assignment within a system of n variables F_n is the action $\mathcal{L}$ of the proof calculus itself **OR** an action B which is logically equivalent to $\mathcal{L}$. Any assignment $\dashv$ not forced by such an action at some point in time t cannot be forced by any other means at t.*[26]

The mathematical essence of this axiom simply implies that, if any program/algorithm, for some given parameters (input), is associated with a trajectory in some space of executions, and thus, a notion of a derivative, then, our calculus can compute the derivative of that derivative, at any point in time.

We will use this argument as the reader will infer, to support the fact that any variables left unassigned by the calculus at any point in time t_i, have no physical, and thus, logical precedent, for being assigned a truth value by any other (mathematical) truth system at t_i.

9.2.6 Termination Of Simple Logic Programs - Partial Functions

The halting problem for a simple horn clause logic program LP w.r.t a partial function simply asks if LP will terminate given a partial input I.
For partial functions, we can still *infer* halting behavior from Expr. 10 *if we force assignments*, as a consequence of Theorem. 9.5 below:

Define **Halt(*F(x)*, H)** as an extension, via a proof system $\mathcal{A}$, of *HaltTot(x, H)* as defined in Expr. 10, where:

- $E \coloneqq x \wedge H$.
- Λ is a 3-SAT transformation of E.
- *F(x)* is the modified 3-SAT formula computed as a result of applying a 3-SAT assignment forcing model to Λ, that is, integrating E.

Theorem 9.5. *For a simple horn clause logic program LP and a partial input ψ, if we ignore both self-referential and self-contradictory function inputs, then we can determine the halting behavior of LP from **Halt(F(LP), ψ)**.*

[25] Our system is a quantum system as discussed in Section. 3.4.6. We cannot directly measure energy along the transverse manifold M_T since we are measuring time, its Pontryagin Dual, which maps energy (total capacity to do work) as measured, at each point in time, by an ambient manifold M_A, to the circle group of displacements in Euclidean space — assignments as infinitely separated antipodes — see [41] for an analogous concept from the field of physics itself. Therefore, we measure only differentials in energy. E is our "quantum" Lagrangian. From this analogy, we can also make a geometric inference — that the integration operation in the calculus is over a *twisted*, infinitely large, (celestial) sphere: $\int_{T(S)} F\,dt$.

[26] We say $\dashv$, by virtue of its inertia – which obeys a form of Mach's principle as well as the Equivalence Principle in physics — is at rest at time t w.r.t to any mathematical system.

Proof. If neither an assignment ρ_i or its negation $\neg\rho_i$ is forced by a set of assignments ψ to *F(LP)*, then we cannot determine if a horn clause $\mathcal{H}_i$ in LP which consists of disjunctions of either negative or positive $\{\rho_i\}$ elements is satisfied or not by ψ and this, by Axiom. 9.1 $\implies$ LP does not halt on ψ.
If, however, for each $\mathcal{H}_i$, which consists of disjunctions of either negative or positive $\{\rho_i\}$, any of the following hold:

- A positive goal is true, **OR**:
- At least one negatively stated premise $\neg\rho_i$ is false and so the goal is true, **OR**:
- Every negatively stated premise $\neg\rho_i$ is true and so every premise ρ_i is false so that the goal becomes false.

$\implies$ LP halts on ψ.
Therefore, if we ignore both self-referential and self-contradictory function inputs, then we can determine the halting behavior of LP from **Halt(*F(LP)*, ψ)**. □

Remark 9.6. *TM-1 cannot make this determination.*

9.2.7 Solving The Halting Problem

We can now reformulate the full classic halting problem, with self-referential and self-contradictory function inputs, in terms of **Halt(*F(LP)*, ψ)**, along with a demonstration that violates the usual paradox-inducing, self-referential assumption used in Section 9.2.1. We show that this assumption, stemming from the even more basic assumption that all computable functions are enumerable, that is, correspond to $\aleph_0$ (Church Turing thesis), does not hold in our system. We will achieve this utilizing 4 key provisions:

Function Ordering The halting function **Halt(*F(x)*, H)** in our calculus can only be computed by the proof calculus itself. It is a higher order function than any λ-function.

Input Restriction The calculus will ***never*** allow any computable function (real-valued λ-function) access to the instance of the *halting function* as an input because the *halting function* is of a higher order, as indicated above. This comes from all programs being expressed formulaically. This rules out other types of problematic inputs as a result.

Forcing Model Any allowed program-input pair (x, H) passed to the calculus will be solved as a 3-*SAT assignment forcing problem* via **Halt(*F(x)*, H)**.

Axiom Of exhaustive Force We invoke the axiom of exhaustive force where necessary and applicable, to infer that the calculus, at that point, has enough information to make a halting decision for allowed inputs.

Theorem 9.6. *For any simple horn clause logic program P, the proof calculus computes the Halting Problem **Halt(**F**(P), I)**, for any input I for which the calculus* **ALLOWS** *execution of P.*

Proof. We prove by contradiction: Assume $\exists g$, where g is a *partial function* as described in Section. 9.2.1, that contradicts the halting decision made by **Halt(**F***(P)*, H)** and where g is computable by P or a by subprogram/enumerable program of P, then:

1. If *Source(Tur(k))*, the source code (horn clause representation) of a program execution *Tur(k)* enumerable in P, which computes a function k, is satisfiable, then $\exists\lambda(k)$ where $\lambda(k)$ is a λ-function computable by the calculus and $\lambda(k)$ computes a satisfying solution for *Source(Tur(k))* as a subformula of the 3-SAT formatted F**(P)**. This also implies that the calculus can compute a forcing model for *Source(Tur(k))*.
2. g can only contradict the decision of **Halt(**F***(P)*, H)** for its own source code *Source(g)* by encoding a contradiction in *Source(g)* in any of the following 2 ways:

(a) g creates a conjunction of its own (3-SAT representation) H with its negation, $\neg H$ to create $\bar{H}$ so that Source(g) $\coloneqq \bar{H}$. This has the following remediation:

i. *Source(g)* is unsatisfiable and so if it is a subformula of some *Source(Tur(g))*, which computes it, then *Source(Tur(g))* cannot be satisfiable.

ii. If *Source(Tur(g))* is unsatisfiable, then for any I, **Halt(F(*Source(Tur(g))*), I)** returns 0 and so g is not contradicted.

iii. If *Source(Tur(g))* is satisfiable, then g cannot create $\bar{H}$, and *THEN*:

iv. For any I, **Halt(*Source(Tur(g))*, I)** returns the correct result.

v. Therefore, g cannot contradict **Halt(F(P), H)** for any allowed input, I.

(b) g creates a non-linear goal such as those used by Devienne et. al [42] where atomic formulas link to each other via **unpredictable** iterations of periodically (piecewise) linear functions as defined by Conway [43]. This has the summarized effect that:

i. g **always** sets up a goal α which is the premise of another function k and:

ii. k **always** sets up a goal which is the premise of another function l where:

iii. l is either **always** g or is **always** linked in a series of such *goal-to-premise* associations to a goal β which is a premise in g.[27]

(c) (2b) above has the following remediation that for any I :

i. if *Source(Tur(g))* does not *truly* halt on I, then:

A. **Halt(F(*Source(Tur(g))*), I)** will never force α to take on a positive value of true since that would mean g is satisfiable.

B. This means neither $\neg\beta$ or any of the other negatively stated premises $\neg\rho_i$ of g will ever be forced to take on a negative value of false since that would mean g is satisfiable and thus, halts. In fact, $\neg\beta$ and at least one $\neg\rho_i$ will *never* be forced to take on any value.

C. If neither of the 2 conditions, 2(c)iA or 2(c)iB, ever occurs, then *Source(Tur(g))* will never be satisfied by I and so
Halt(F(*Source(Tur(g))*), I) always returns the correct answer of 0, that is, that *Source(Tur(g))* never halts on I, by invoking Axiom. 9.1.

ii. For any function f that directly recurses on (directly calls) itself, the algorithm can simply check if there is some fixed term for which the recursion terminates since f is now being computed as a fixed-point for such a recursion at every point in time.

iii. If either of the 2 conditions, 2(c)iA or 2(c)iB, ever occurs, then *Source(Tur(g))* will halt on I) and so
Halt(F(*Source(Tur(g))*), I) always returns the correct answer of 1.

The proof calculus can, after applying a forcing model, **predict/infer** via the above arguments, if any simple horn clause logic program P halts on any input I for which the calculus ***ALLOWS*** an execution of P. □

Theorem 9.7. *The computational model* Γ *of* ***Halt(F(P), H)*** *can compute the standard halting function.*

Proof. Since Γ has full *SOL* quantificational scope, it can decide for any program-input pair $\{P, I\}$, if $\{P, I\}$ halts by checking any other program input pairs that form a computational sequence (finite or infinite) with $\{P, I\}$. □

[27] [42] shows that these conditions can be induced in a logic program with just one horn cause if we allow premises themselves to be complex so that atomic formulas can have *intermediate* internal goals (representable in execution, as horn clauses with just a goal(s), and no premises) as elements of the relation. Since relations can be written out as conjunctions and so become disjunctive when incorporated in an outer horn clause, any *implicit* inner horn clauses will become conjunctive in expanded form. And so, we can have outer horn clauses that can be converted to full k-SAT formulas. Of course, our argument in the proof stays the same for this "higher-ordered" horn clauses, as the resulting formula can always be converted to the 3-SAT format.

Halt(*F(P)*, *H*) is a $2^{\aleph_0}$-function since it has the cardinality of ξ, used in Definition. 9.1, shown to be $2^{\aleph_0}$ (every 3-SAT formula corresponds to a unique **Halt(*F*, *H*)**). This demonstrates that $2^{\aleph_0}$-functions are *provably* computable, thus, violating the set-theoretic implication of the Church-Turing thesis.

Remark 9.7. *Our demonstration suggests that a good proof of $P \neq NP$ with the assumption provided in the Church-Turing Thesis should be able to show that a deterministic algorithm for an NP problem is uncomputable according to that thesis. In fact, if the approach in this paper is ever disproven on any grounds, then this aspect of the demonstration and the technique used may be the sole surviving aspect of the paper worth preserving in the literature. We demonstrate such a $P \neq NP$ proof below.*

Lemma 9.2. *For an algorithm A to run in polynomial time on any arbitrary 3-SAT formula, it cannot backtrack over more than ε number of variables for any given given formula F, where ε is a constant independent of the number of variables in F.*

Proof. If A cannot provide such a guarantee for any formula, then it must necessarily backtrack over $c * n$ number of variables in the worst-case, where $c \in \mathbb{Q}$ and n is the number of variables in the formula. Therefore, A must necessarily run in exponential time in the worst-case. Therefore, for algorithm A to run in polynomial time on any arbitrary 3-SAT formula F, it cannot backtrack over more than a fixed, constant ε number of variables in F. □

Lets us define ε as the *Polynomial Error Limit*.

Lemma 9.3. *Any algorithm A with a* Polynomial Error Limit *implements Axiom. 9.1 for at least $n - \varepsilon$ number of variables in a formula F with n variables.*

Proof. If A does not implement the Axiom. 9.1 on at least $n - \varepsilon$ number of variables in F, then it will necessarily backtrack over $> \varepsilon$ number of variables, invalidating the assumption that A has a *Polynomial Error Limit*. □

Theorem 9.8. *Any algorithm A that encodes a* Polynomial Error Limit *can be used to solve the halting problem for inputs of at most $n - \varepsilon$ number of variables in any formula F.*

Proof. This follows from Theorem. 9.7 where we can view Γ as having a *Polynomial Error Limit* of 0. □

Remark 9.8. *This argument is based on a computability argument - reducing* P = NP *to the halting problem - rather than a complexity-theoretic argument. This fact already exempts our approach from the normal barriers that normally limit such complexity-based arguments in how they can be used to* separate *complexity classes.*

9.2.8 Computability From Spacetime Curvature

We understand *TM-2*, which orders **Halt(*F(P)*, *H*)**, in its most general case as the ordering imposed by the Second Order predicate of motion indicated in Expr. 4. Here, we seek to briefly get a perspective on the *logic of the computation* of the predicate itself as it derives from the geometric properties of the computational space.

We examine another core result in the theory of computability literature that generalizes the classical halting problem, for hints about the geometric properties of this logic - Rice's theorem. Rice's theorem is summarily described with the following definitions:

Definition 9.7. *A non-trivial property is **defined** as a property which is neither true for* every program*, nor false for* every program*, in some system, for example, a Turing Machine.*

Definition 9.8. *A semantic property is **defined** as a mathematically expressible statement of a property about a program's behavior, for example, its termination behavior on any input.*

Rice's Theorem then states that, *all non-trivial semantic properties of programs are undecidable.* We can immediately see that our proof calculus violates this theorem because of its temporal model, which differentially modifies the predicate of motion at each point in time in the underlying second-order model of computation. The only trivial point (which applies to *all* programs) in our meta-program is at time $t = 0$.

The term, *every program*, in the definition of Rice's theorem, then becomes the distinguishing term for our analysis — because at later times (than $t = 0$), some programs no longer belong to the current scope of quantification, as they have been complexly differentiated out. In essence, we compute the predicate in real-valued time but differentiate it in complex-valued time (see A.1). Our second-order logic then is no longer a flat spatial logic, but rather a curved space-time logic. Curvature here implies that we have torsion, a *measure* of the failure of the differentiating trajectory to be contained in a *fixed* plane. As in the case with the curved spacetime used explicitly in General Relativity in physics, each point of (spatial) inference P, comes bundled with a temporal component which induces a Riemannian metric (represented as an inner tensor product) on P as the tangent vector of motion.

Also, as a consequence of the stated *curvature* of the logic, we now only have ordering in *spacetime* and not in space (without time) alone (apart from the trivial least fixed-point). We can always add a space ordering if we traverse all possible solutions in an ordered way. In this way our logic in the form presented is comparable to the first order descriptive complexity approach of extending unordered first order logic by an induction operator [44].

While we plan to explore the full details of this geometric implication in future work, one can readily extend the approach taken in this work to other uncomputable problems on the basis of our short analysis here. This can be done by combining information about the geometric nature of the implications (curvature) in the form stated here with the notion of *properties of a program* as described in Rice's theorem. Just as the halting property was correlated, one could map some uncomputable problem to some other time-dependent property of a program. This will typically involve marking the start and end states with some special mathematical meaning and then using some kind of *reachability* argument for some end state Ω mapped to some property Ψ. Such an argument would aim to construct a *T-Schema*, in the sense used by Tarski, for proving the truth of Ψ:

$$(\Psi \text{ is true}) := \top \iff \exists . \mathit{Reachable}\,(\Omega)$$

With this kind of approach, one could even begin to re-examine other mathematical notions like general provability (Gödel's theorems) from this point of view (of course, this is dependent on a deeper understanding of the arithmetical properties of underlying sets, as discussed earlier). For this it may be helpful to understand other models of computation similar to the one we explored here — for example those that use categorical semantics at some level of computation, such as the notion of monoidal computers/string diagrammatic computation, introduced by Pavlovic in a series of papers beginning with [45] — or those that take the idea of automatic differentiation to the level of computing derivatives for Turing Machines and other discretely valued algorithms [46] — or even, looking at hypothetical models outside of the mathematics/computer science models, that suggest new computational substrates, such as a quantum *field* computer [47], which we deem as closest to the model we espouse here – with computations possibly governed by evolutionary algebraic structures and operations [48].

One possible outcome of the activity suggested is that, upon fully understanding the arithmetic, logical and geometric properties of the computational space used, one could show that it corresponds wholly to actual possible mathematical universes (of possible sets and their associated functions). In such a case, one could easily show, for example, that the Continuum Hypothesis is true, since no other sets of objects exist, in this constructible Λ-*universe*, between the set of possible assignments of cardinality ($\aleph_0$) and the set of satisfying (λ) functions of cardinality ($2^{\aleph_0}$). This would be so if well-orderings of $\aleph_0$ was shown to be isomorphic to *TM-2*, the temporal ordering of *DetAlg*s. Of course, a deeper understanding could also always point in the opposite direction.

10 Summary

We have presented a proof calculus for the 3-SAT problem via a description of an algorithmic system which implements an Operator-Valued Algebraic Second-Order Logic, with a Relative Least Fixed-Point [A-SO-RLFP]. We have also discussed some implications of this logic for computation.

We plan to extend this effort, in future work, from one which implements a framework of computation as (logical) inference, to one of which comprehends computation as (logical) *measurement*. This will involve understanding the deeper mathematical as well as physical underpinnings of the results presented. Indeed, most of the topological and geometric intuitions used in the paper is inspired, as well as sourced, from the Holographic Principle [49] in physics, as well as some its theoretical realizations, particularly the AdS/CFT correspondence [50].

We also see potential applications for the approach presented here, for the field of Artificial Intelligence, in that we have broached upon the subject of meta-computation, which encodes knowledge (cognition) about computation (symbolic machine reasoning modeling actually being the original motivation behind the authors research efforts).

References

[1] *V*aliant, Leslie (17–19 October 2004). Holographic Algorithms (Extended Abstract). FOCS 2004. Rome, Italy: IEEE Computer Society. pp. 306–315. doi:10.1109/FOCS.2004.34. ISBN 0-7695-2228-9.

[2] *J.* -Y. Cai, V. Choudhary and P. Lu (2007). On the Theory of Matchgate Computations. Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07), San Diego, CA, USA, 2007, pp. 305-318, doi: 10.1109/CCC.2007.22.

[3] *S*tephen A. Cook. The complexity of theorem-proving procedures (1971). In Proceedings of the 3rd ACM Symposium Theory of Computing, pages 151–158, Shaker Heights, Ohio, 1971.

[4] *L*eonid A. Levin. Universal search problems (1972). Problemy Peredachi Informatsii, 9(3):265–266, 1973.

[5] *R*ichard M. Karp. Reducibility among combinatorial problems. Complexity of Computer Computations, (R. Miller, J. Thatcher eds.), pages 85–103, 1972.

[6] *C*ai, J. (2010). Holographic algorithms: From art to science. Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC), 401–410. https://doi.org/10.1145/1806689.1806746.

[7] *C*ai, J., Lu, P., and Xia, M. (2009). Holographic algorithms by Fibonacci gates. Computational Complexity, 18(4), 598–642. https://doi.org/10.1007/s00037-009-0281-9.

[8] *L*andsberg, J. M., Morton, J., and Norine, S. (2009). Holographic algorithms with matchgates capture precisely tractable planar #CSP. SIAM Journal on Computing, 40(3), 699–742. https://doi.org/10.1137/090770064.

[9] *L*andsberg, J. M., Morton, J., and Norine, S. (2013). Holographic algorithms without matchgates. Linear Algebra and its Applications, 430(2–3), 314–344. https://doi.org/10.1016/j.laa.2008.09.014.

[10] *C*ai, J., Lu, P., and Xia, M. (2009). Holographic reduction, interpolation and hardness. Computational Complexity, 18(2), 238–278. https://doi.org/10.1007/s00037-009-0277-5.

[11] *C*ai, J., Lu, P., and Xia, M. (2009). Classification of a class of counting problems using holographic reductions. SIAM Journal on Computing, 38(5), 1674–1701. https://doi.org/10.1137/070708113.

[12] *B*evers, E. and Lewi, J. (1991). Proof by consistency in conditional equational theories. In S. Kaplan and M. Okada (Eds.), Conditional and Typed Rewriting Systems (CTRS 1990). Lecture Notes in Computer Science, vol. 516. Springer–Verlag. DOI: 10.1007/3-540-54317-1

[13] *W*inkler, F. (1989). Equational Theorem Proving and Rewrite Rule Systems. In J. Retti and K. Leidlmair (Eds.), 5. Österreichische Artificial-Intelligence-Tagung, Informatik-Fachberichte, vol. 208. Springer–Verlag. DOI: 10.1007/978-3-642-74688-8_3

[14] *B*ouhoula, A., Jouannaud, J.-P., and Meseguer, J. (2000). Specification and proof in membership equational logic. Theoretical Computer Science, 236, 35–132. (Earlier versions: TAPSOFT'97, LNCS 1214, 1997.)

[15] *U*rquhart, A. (2010). Chapter: An equational calculus in Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press.

[16] *B*eame, P., Impagliazzo, R., Krajíček, J., Pitassi, T., and Pudlák, P. (1996). Lower bounds on Hilbert's Nullstellensatz and propositional proofs. Proceedings of the London Mathematical Society (3) 73:1–26.

[16] *C*legg, M., Edmonds, J., and Impagliazzo, R. (1996). Using the Gröbner basis algorithm to find proofs of unsatisfiability. STOC 1996, 174–183.

[17] *I*mpagliazzo, R., Pudlák, P., and Sgall, J. (1999). Lower bounds for the Polynomial Calculus and the Gröbner basis algorithm. Computational Complexity, 8(2):127–144.

[18] *G*rigoriev, D. (1998). Tseitin's tautologies and lower bounds for Nullstellensatz proofs. FOCS 1998, 648–652. Math Logic and Discrete Math.

[19] *A*lekhnovich, M. and Razborov, A. A. (2001). Lower bounds for Polynomial Calculus: Non-binomial case. FOCS 2001.

[20] *G*rigoriev, D. and Vorobjov, N. (2001). Complexity of Null- and Positivstellensatz proofs. Theoretical Computer Science / related proceedings.

[21] *D*owling, William F.; Gallier, Jean H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. Journal of Logic Programming, 1 (3): 267–284, doi:10.1016/0743-1066(84)90014-1, MR 0770156.

[22] *H.* Zhang and M. Stickel (1996). An efficient algorithm for unit-propagation. Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics.

[23] *L*ee A. and Armin Biere (2021). Non-clausal Redundancy Properties. Automated Deduction – CADE 28, Lecture Notes in Computer Science, 2021.

[24] *R.*E. Wengert (1964). A simple automatic derivative evaluation program. Comm. ACM. 7 (8): 463–464. doi:10.1145/355586.364791. S2CID 24039274.

[25] *B*aydin, A. G., Pearlmutter, B. A., Radul, A. A. and Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research, 18(153), 1–43.

[26] *H*alim, M. A. S. B. and Hoe, T. T. (2020). Introduction to Automatic Differentiation and Neural Differentiation Equation. Proceedings of the International Conference on Science and Mathematics, pp. 81–89.

[27] *W*ang, L. and Zhao, J. (2023). Algorithmic Differentiation. Architecture of Advanced Numerical Analysis Systems, pp. 49–85. Apress, Berkeley, CA. DOI : 10.1007/978-1-4842-8853-5_3.

[28] *R*ehner Philipp and Bauer Gernot. (2021) Application of Generalized (Hyper-) Dual Numbers in Equation of State Modeling. Frontiers in Chemical Engineering Volume 3 https://www.frontiersin.org/articles/10.3389/fceng.2021.758090 DOI : 10.3389/fceng.2021.758090 ISSN=2673-2718.

[29] *A*lexander Beilinson, Vladimir Drinfeld (2004). Chiral Algebras, Colloquium Publications 51, Amer. Math. Soc. 2004.

[30] *G*irard, Jean-Yves (1989). Geometry of interaction 1: Interpretation of System F. Studies in Logic and the Foundations of Mathematics. 127: 221–260.

[31] *G*irard, Jean-Yves (2011). Geometry of interaction V: Logic in the Hyperfinite Factor. Theoretical Computer Science. 412 (20): 1860–1883.

[32] *M*artin Davis and Hilary Putnam (1968). A Computing Procedure for Quantification Theory. J. ACM. 7 (3): 201–215. doi:10.1145/321033.321034. S2CID 31888376. p. 210, "III. Rule for Eliminating Atomic Formulas".

[33] *R*obinson, J. Alan (1965). "A Machine-Oriented Logic Based on the Resolution Principle". Journal of the ACM. 12 (1): 23–41. doi:10.1145/321250.321253. S2CID 14389185.

[34] *S*amuel R. Buss (1998). Lower bounds on Nullstellensatz proofs via designs. Proof Complexity and Feasible Arithmetics, volume 39 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 59–71. American Mathematical Society, Providence, RI, 1998.

[35] *G*rothendieck, Alexander (1957). Sur quelques points d'algèbre homologique. Tohoku Mathematical Journal, Second Series, 9: 119–221, doi:10.2748/tmj/1178244839, ISSN 0040-8735, MR 0102537.

[36] *T*ärnlund, S.Å. (1977). Horn clause computability. BIT Numerical Mathematics. 17 (2): 215–226. doi:10.1007/BF01932293. S2CID 32577496.

[37] *A*ndréka, H.; Németi, I. (1978). The generalised completeness of Horn predicate-logic as a programming language. Acta Cybernetica. 4 (1): 3–10.

[38] *R*. Penrose (1967). Twistor Algebra. Journal of Mathematical Physics, vol. 8, no. 2, pp. 345–366, 1967. DOI: 10.1063/1.1705200.

[39] *R*. Penrose (1968). Twistor Quantisation and Curved Space-Time. International Journal of Theoretical Physics, vol. 1, no. 1, pp. 61–99, 1968. DOI: 10.1007/BF00668831.

[40] *R*. Penrose (1969). Solutions of the Zero-Rest-Mass Equations. Journal of Mathematical Physics, vol. 10, no. 1, pp. 38–39, 1969. DOI: 10.1063/1.1664756.

[41] *S*. Pasterski, M. Pate, and A.-M. Raclariu, "Celestial Holography" in 2022 Snowmass Summer Study. 11, 2021. arXiv:2111.11392 [hep-th].

[42] *D*evienne, P., Lebègue, P., and Routier, J.-C. (1993).Halting problem of one binary Horn clause is undecidable. Lecture Notes in Computer Science, vol. 665, pp. 48–57.

[43] Conway, J.H. (1972). Unpredictable Iterations. Proceedings of the 1972 Number Theory Conference, University of Colorado, Boulder, pp. 49–52.

[44] *D*awar, Anuj (1993). Feasible computation through model theory. University of Pennsylvania ProQuest Dissertations Publishing, 1993. 9321378.

[45] *D*usko Pavlovic (2013). Monoidal computer I: Basic Computability By String Diagrams. Information and Computation, 222:171–195, 2013. doi:10.1016/j.ic.2012.12.003.

[46] *J*ames Clift and Daniel Murfet. Derivatives of Turing machines in linear logic. arXiv preprint arXiv:1805.11813, 2019.

[47] *F*reedman M. H (1998). P/NP, and the quantum field computer. Proc Natl Acad Sci USA 95 (1), 98-101 (1998).

[48] *J*oyal, André (October 1981). "Une théorie combinatoire des séries formelles". Advances in Mathematics. 42 (1): 1–82. doi:10.1016/0001-8708(81)90052-9.

[49] *B*ousso, Raphael (2002). The Holographic Principle. Reviews of Modern Physics. 74 (3): 825–874.

[50] *J*. M. Maldacena (1998). "The Large N limit of superconformal field theories and supergravity," Adv. Theor. Math. Phys. 2 (1998) 231–252.

A Appendix

Theorem A.1. *Define ω as the metric on $M(\sigma)$, the space of (negative) motions of σ-functions, then $\omega \in \mathbb{C}$.*

Proof. We assume, for clarity's sake, that we are working with an underlying formula with more than one solution. Any results obtained should automatically apply to formulas with a single solution.
We analyze the values of displacement caused by motions of functions:

- Each forward motion of a λ-function trajectory is antipodal to its complement (the set of excluded directions), that is truth assignments to distinct variables , as functions, "move" in opposite directions.
- Each forward motion requires a plane of motion.
- The planar component of the motion at time $t = i$ is the norm that induces the metric associated with the motion. This is exactly the set of negatively constrained (disallowed) motions (to internal nodes of the solution space) at time $t = i$, encoded as the constraint function.
- We previously denoted the total measure on the space (of all possible motions) at any point in time, as the total probability measure 1. This measure is evenly distributed among all possible motions/trajectories of the λ-function.
- The set of negatively constrained motions θ *must* also have a non-zero measure, $P(\sigma)$, since θ is a non-empty set.
- $P(\sigma)$ *must* have a planar numeric value, as a measure (Item No. A).
- $P(\sigma)$ must have a negative value, relative to the one assigned to the set of positive motions. We assign this value to -1.
- The above assignment to -1 is consistent with the fact that:
  - If all motions were reversed and we denoted the set of negatively constrained motions as positive motions by:
  - Rerecording each starting unassignable statement recorded to a constraint function of a clause at time $t = 0$ as a conjunction of the negations of all the (previously) disallowed truth assignments, for example, we reversed '2,-4,9' to '-2,4,-9'.
  - Then, the value of $P(\sigma)$ will be 1 in the positive direction indicated by the new implied formula.
- The number of elements contained in the set of negative constraints increases by a power of 2 at each point in time.
- If the present value of $P(\sigma)$ is -1 at time t, its previous value at time t - *1* must be the imaginary unit i - this is consistent with the designation of topological multiplication as elimination (here, of increasing imaginaries by powers of 2). This is a type of *complex multiplication.*
- The displacement measured in $M(\sigma)$, the metric space of negative motions, corresponds to the absorbed imaginary value.
- Therefore the metric, ω on $M(\sigma)$ always has both an imaginary part ϕ_i, as well as a real part ψ_r where ψ_r is measured as the negative displacement of ω, along the real line of λ, away from the final λ output, which is $\in \mathbb{R}$ (from Corollary. 9.1 that *0*-form lambda functions are real-valued)
- Therefore, $\omega \in \mathbb{C}$.

□

Note: ω is the metric associated with the (time-valued) inputs and outputs of $\delta(t)$-functions, as well as the (space-valued) outputs (codomain) of σ-functions. σ-functions share a domain with λ-functions.

Theorem A.2. *λ-functions are real-valued functions: $\mathbb{R} \to \mathbb{R}$.*

Proof. Define α as the metric on $M(\lambda)$, the space of (positive) motions of λ-functions.

- From Theorem. A.1, the metric ω at very point in time t on $M(\sigma)$ is complex.
- Since negative motions are eliminations of imaginaries, the pointwise real value decrement of ω must be the additive inverse of the incremented value of the associated positive motion of the λ-function, at the point p_t on the trajectory of motion, corresponding to t.
- This real value increment at p_t is α.
- Thus, $\alpha \in \mathbb{R}$.
- α is the metric associated to each pointwise output of a λ-function, which where applicable, is the next input.
- The input at time $t = 0$ and the the output at time $t = n$ (for a system of n variables, $F(\mathrm{n})$) of a λ-function are $\in \mathbb{R}$ - Corollary. 9.1.

Therefore, all λ-functions, at any point in time, are real-valued: $\mathbb{R} \to \mathbb{R}$. □

B Appendix

Subroutine Definitions. The Definitions below allow implementing algorithms to use other notation other than the standard DIMACS notation we have used in the paper. Therefore, we state unassignable statements and witnesses in their clausal form to facilitate a generic and implementation-agnostic description. The reader can consult the relevant sections of the paper to determine how to use the DIMACS notation (with the 'minus' for negative literals and 'no sign' for positive literals) to implement these routines.

Definition B.1 (Clause-to-Witness Conversion)**.** *Given a clause $C = (\ell_1 \vee \ell_2 \vee \ell_3)$ where each ℓ_i is a literal (over $\{x_1, \ldots, x_n\}$), define*

$$\textsc{ClauseToWitness}(C) \;=\; \{\neg\ell_1, \neg\ell_2, \neg\ell_3\}.$$

That is, a clause is converted to an initial unassignable statement *consisting of the negations of its literals. This encodes the condition that the clause is unsatisfied only when all its literals are false simultaneously.*

Definition B.2 (Witness Correlation)**.** *Let d, s be witnesses, each a finite set of literals with $|d|, |s| \leq B$. Suppose d and s contain exactly one conflicting pair $(\ell, \neg\ell)$. Define*

$$\textsc{Correlate}(d, s) \;=\; (d \cup s) \setminus \{\ell, \neg\ell\}.$$

That is, we form the union of d and s and cancel the conflicting literal pair. The result is a new witness r of size at most $|d| + |s| - 2$. If no unique conflicting literal exists, the procedure is undefined.

Definition B.3 (Witness Reduction)**.** *Let d be a witness and a a truth assignment to some variable v. Define*

$$\textsc{Reduce}(d, a) \;=\; \begin{cases} \bot & \text{if some literal } \ell \in d,\ \ell \coloneqq \neg a, \\ d \setminus \ell, \ell \in d, \ell \coloneqq a & \text{otherwise.} \end{cases}$$

Here $\bot$ denotes that the witness is eliminated (since it is no longer required). If d ever reduces to the empty set $\emptyset$, this indicates a contradiction.

Definition B.4 (Check Blocking Scenarios 2 - 4)**.** *CheckBlocking(S, a)* =

Input: A Differential Form Closure S of unassignable statements.
Input: A target assignment a.
Output: Either 'True' if a causes any Blocking Scenario 2 - 4 or 'False' otherwise.

Initialization:
$Z \leftarrow \emptyset$; `// Set of forced assignments`
foreach $s \in S, |s| \leq 2$ **do**
  $s' \leftarrow$ CopyOf(s);
  $z \leftarrow$ Reduce(s', a);
  if $|z| = 1$ **then**
    if $\exists \neg z \in Z$ **then**
      return 'True'
    end
    else
      insert z into Z;
    end
  end
  return 'False'
end

Algorithm 2: Check Blocking Scenarios 2 - 4