Analogico-Deductive Generation of Gödel's First Incompleteness Theorem from the Liar Paradox*

John Licato, Naveen Sundar Govindarajulu,
Selmer Bringsjord, Michael Pomeranz, Logan Gittelson
Rensselaer Polytechnic Institute (RPI)
Troy NY USA
{licatj,govinn,selmer,pomerm,gittel}@rpi.edu

Abstract

Gödel's proof of his famous first incompleteness theorem (**G1**) has quite understandably long been a tantalizing target for those wanting to engineer impressively intelligent computational systems. After all, in establishing **G1**, Gödel did something that by any metric must be classified as stunningly intelligent. We observe that it has long been understood that there is some sort of analogical relationship between the Liar Paradox (**LP**) and **G1**, and that Gödel himself appreciated and exploited the relationship. Yet the exact nature of the relationship has hitherto not been uncovered, by which we mean that the following question has not been answered: Given a description of **LP**, and the suspicion that it may somehow be used by a suitably programmed computing machine to find a proof of the incompleteness of Peano Arithmetic, can such a machine, provided this description as input, produce as output a complete and verifiably correct proof of **G1**? In this paper, we summarize engineering that entails an affirmative answer to this question. Our approach uses what we call *analogico-deductive reasoning* (ADR), which combines analogical and deductive reasoning to produce a full deductive proof of **G1** from **LP**. Our engineering uses a form of ADR based on our META-R system, and a connection between the Liar Sentence in **LP** and Gödel's Fixed Point Lemma, from which **G1** follows quickly.

1 Introduction

Gödel's proofs of his incompleteness theorems are among the greatest intellectual achievements of the 20th century. Even armed with the suggestion that the Liar Paradox (**LP**) might somehow serve as a guide to proving the incompleteness of Peano Arithmetic (**PA**),[1] the level of creativity and philosophical clarity required to actually tie the two concepts together and produce a valid proof is staggering; it certainly should not be controversial to claim that no computational reasoning system can, at present, achieve this sort of feat without significant human assistance.

1.1 Automating the Proof of G1

Prior work devoted to producing computational systems able to prove **G1** have yielded systems that manage to prove this theorem only when the distance between this result and the starting point is quite small. This for example holds for the first (and certainly seminal) foray; i.e., for [**?**], as explained in [**?**], where it's shown that the proof of **G1**, because the set of premises includes an ingenious human-devised encoding scheme, is very easy—to the point of being at the level of proofs requested from students in introductory mathematical logic classes.

Likewise, [**?**] is an exact parallel of the human-devised proof given by [**?**]. Finally, in much more recent and truly impressive work by [**?**], there is a move to natural-deduction formats, which we applaud—but the machine essentially begins its processing at a point exceedingly close to where it needs to end up. As Sieg and Field concede: "As axioms we take for granted the representability and derivability conditions for the central syntactic notions as well as the diagonal lemma for constructing self-referential sentences." If one takes for granted such things, finding a proof of **G1** is effortless for a computing machine.[2] In sum, while a lot of commendable work has been done to build the foundation for our prospective work, the daunting formal and engineering challenge of producing a computational system able to produce **G1** without clever seeding from a human remains entirely unmet.

2 The Analogico-Deductive Approach

2.1 Conjecture Generation

The problem with the purely deductive method is simply that it does not allow us to come close to the type of model-based reasoning that great thinkers are known to have used. Gödel himself has been described as having a "line

*We are deeply grateful for penetrating feedback provided by three anonymous referees, and for financial support from AFOSR and the John Templeton Foundation.

[1] **G1** of course applies to any axiom system meeting the standard conditions (Turing-decidability, representability, consistency), but we tend to refer to **PA** for economization.

[2] A video demonstration of the small-distance process can be found at http://kryten.mm.rpi.edu/GodelI_abstract_in_Slate.mov.

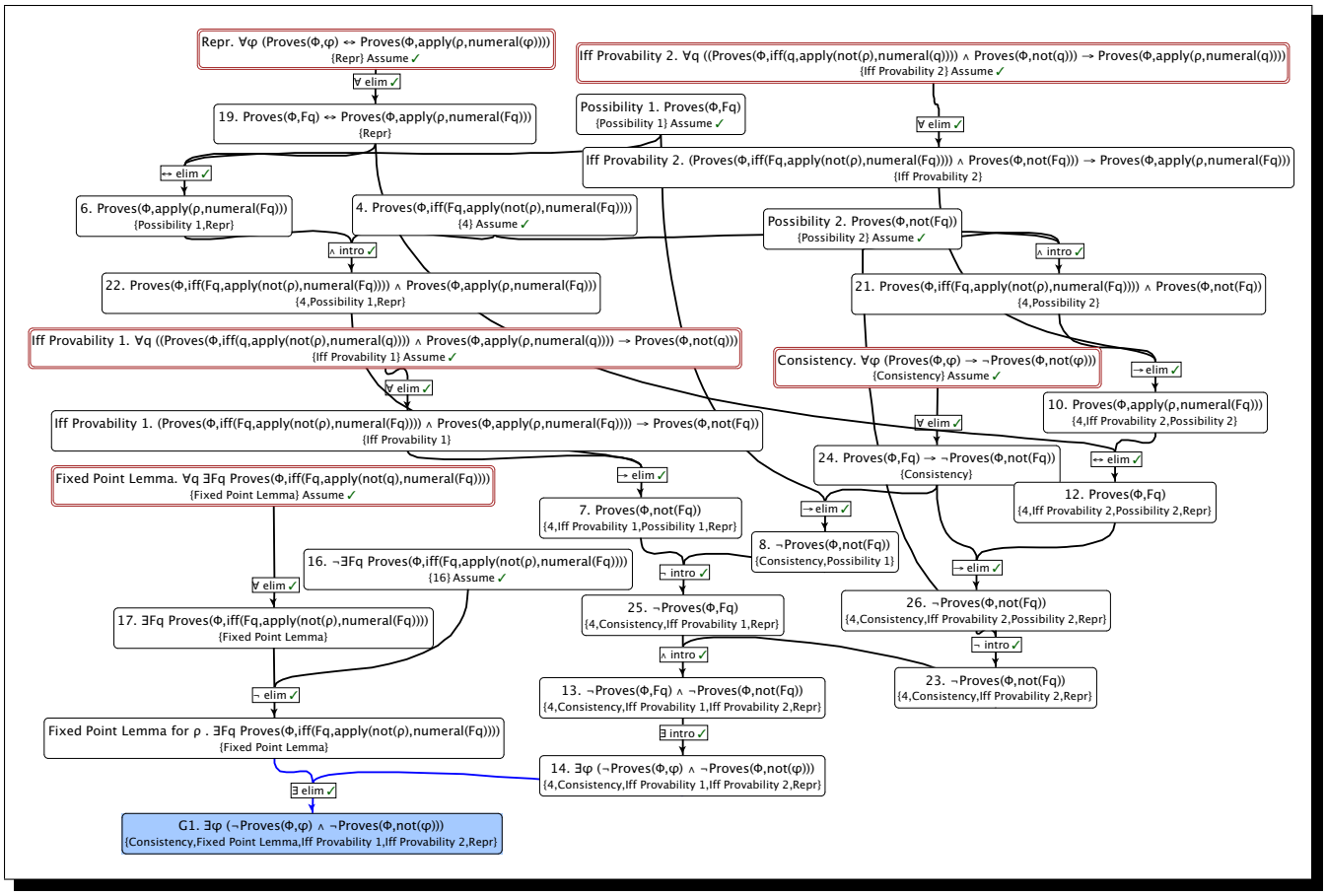

Figure 1: A full deductive "short distance" proof of **G1** in Slate, manually constructed. The final statement (highlighted) can be deduced from the statements in the red boxes. Notation used here is described in Section 3.3.

of thought [which] seems to move from conjecture to conjecture" [**?**]. Reasoners in general are known to conjecture through analogy when a straightforward answer to their current problems cannot be found by other methods [**?**; **?**; **?**]. Analogical reasoning as a tool for conjecture or hypothesis generation, then, can apparently offer new ideas, a strength that is missing from purely deductive approaches.

Since our goal here is a complete deductive proof of **G1** using the model-based reasoning of Gödel, we must make use of a technique that integrates analogical and hypothetico-deductive reasoning. This technique, developed in [**?**] and [**?**], takes as its input source and target domains, and performs analogical mapping and knowledge transfer in order to produce candidate hypotheses in the target domain. On the basis of further deductive reasoning and other validation techniques, these hypotheses are either rejected or accepted.

2.2 The Liar Paradox and G1

The scope of the work we present in this paper should now be clear: We intend to use ADR to produce a complete and verifiably correct proof of **G1**. Our source and target domains are **LP** and **G1**, respectively. However, the transfer of knowledge between the two domains is not a trivial task: Whereas **G1** is completely formal and in a domain dealing with formal theories and well-formed-formulas, descriptions of **LP** are often semi-formal at best, and problems which claim to be equivalent to **LP** have involved objects as diverse as barbers in small towns, knights, knaves, and ancient Cretans. We reiterate here that an informal understanding of **LP** is to be used as no more than a *guide* to the discovery of **G1**. Much more creative thought is required to formulate concepts such as Gödel numbering or primitive-recursive functions.

LP, in its simplest form, is a self-referential statement that at first glance seems cute and unproblematic, but upon closer scrutiny leads to an apparent contradiction. "This statement is false" is a typical version; the problem appears when one tries to determine if the statement is actually true or not. If true, it's false; if false, it's true. Thus, a contradiction appears if we assume that all such statements must have one and exactly one truth value. The Gödel statement, which may be informally stated as "This statement is not provably true," faces a similar problem. If it is provably true, it is false, and thus cannot be provably true; if it is provably false, then the statement is provably true, and this implies that the reasoning system is inconsistent.

2.3 From Conjectures to Proof Generation

Assuming then that we have an analogical reasoning system able to surmount the difficulties of cross-domain matching and translation—even when those domains differ in their level of formality—we next tackle the problem of what to do with the statements that are analogically transferred by the ADR system. Our system is given two collections of statements S and T from the source and target domains respectively, and various pieces of domain knowledge necessary to perform patching operations (which we will describe shortly). T must consist at least of some axioms, and the desired goal statement. The statements in S and T are matched analogically, and on the basis of this match, the statements in S are analogically transferred to become S'. If the source domain is less formal than the target domain, the transferred statements may not be well-formed in the target domain, violating some syntactic rule that exists in the target domain but was not detected by the analogical matcher. This is achieved through the use of what we call *patching operators*.

Patching operators are, by design, meant to fix issues that should be trivial for a human reasoner familiar with the target domain. Although they are an important part of the full automation of the proof-generation process that is our goal, they must necessarily be very limited in their scope in order to prevent over-zealous corrections that could negate the benefits of using analogical transfer in the first place. It is not clear at which point patching operators become more harmful than beneficial; this is the topic of ongoing investigation. Presently, we only implement two such rules:

PO1: The analogically transferred statement s' violates some typing constraint. As an example, consider the source statement $s \equiv Says(p)$, where p is some proposition. Assume the system generates the corresponding target statement $s' \equiv IsValidProof(p)$, where $IsValidProof$ is a predicate that takes a Gödel numeral as its argument. Since p is a proposition and not a numeral, a type constraint is violated. A human reasoner would likely draw on domain-specific knowledge about how to fix this sort of thing, and apply a casting function to produce $s'' \equiv IsValidProof(numeral(p))$. Knowledge of which casting functions might perform this task are assumed to be part of the domain.

PO2: The analogically transferred statement s' violates an arity constraint; e.g., $\phi \equiv (B(p) \wedge B(p))$ as the source statement, and $\phi' \equiv (Proves(\Phi, p') \wedge Proves(\Phi, p'))$ as the target. The arities of the predicates differ, and because B maps to $Proves$ and p to p', a source statement $s \equiv B(p)$ may transfer as $s' \equiv Proves(p')$, which is not well-formed in the target domain. This can be patched by checking examples in which B is mapped to $Proves$, then replacing the missing arguments. Because the system determines how to patch these statements using the analogical mapping (using a best-guess from the knowledge available), it requires no additional domain knowledge.

Patching is applied to all eligible statements in the set of transferred statements S'. Once the patching process is complete, the statements are introduced into the target domain. Those that could not be patched enough to become well-formed in the target domain are discarded. The remaining set of statements, S'', are first partially ordered based on which statements they corresponded to in S, the original proof of **LP**. For every $s_1, s_2 \in S$, if s_2 has s_1 as one of its dependencies in the proof of **LP**, then s_1 comes before s_2 in the partial ordering. It is likewise for their corresponding statements in S''.

In this order, each statement $s \in S''$ is integrated into the target domain, depending on which of the following conditions are met:

1. s exactly matches or is logically equivalent to some axiom a in the target domain. In this case, we discard s.
2. s logically follows from a (possibly empty) subset of the statements in the target domain. Here, we keep s in the target domain, as it may be useful later as a lemma.
3. Some subset A of the axioms in the target domain follow from s. In this case, s is checked to see if it implies a contradiction. If not, it is kept. Otherwise, it is discarded because anything can follow from a contradiction and the axioms in A would follow trivially, which would not be helpful to our ultimate goal. Contradictions may occur from corruptions of the analogical transfer process, which may happen with poor analogical mappings; this fact emphasizes the importance of deductive reasoning in any application of ADR.
4. None of the above. In this case, s is kept as an axiom.

Finally, the system checks if the goal statement of the target domain follows from the resulting statements. If so, the resulting proof is submitted as output.

3 Simulation

The nature of this project requires bringing together many different tools. We describe two major components: META-R for analogical reasoning; and Slate for deductive reasoning, and high-level proof visualization and verification.

3.1 Slate

Slate is a graphical proof-construction environment based on natural deduction, and includes support for constructing proofs in propositional logic, first-order logic, and several modal logics. Slate also has the ability to automatically discover proofs via resolution, by calling ATPs; e.g., SNARK [**?**]. This feature allows one to utilize Slate in a hybrid mode to construct proofs that are semi-automated.[3] Proofs in Slate can be viewed as a directed acyclic graph $\mathcal{G} = \{\langle \mathcal{F}, \mathcal{I} \rangle, E\}$ with two types of nodes: formula nodes $\mathcal{F}$ and inference rule nodes $\mathcal{I}$. Each inference rule node corresponds to an application of the inference rule in the proof and has as parents the premises of the rule, and as children the conclusion of the rule. Each formula node f has a complex structure comprised of a unique identifier id, the formula ϕ corresponding to the node, and a set of identifiers, T, corresponding to the *scope* under which the current formula was derived. More concisely, each formula node $f := \langle \mathsf{id}, \phi, \mathsf{T} \rangle$. To prove a formula ϕ from a set of premises Γ, one needs to construct a graph that has a formula node $f = \langle \mathsf{id}, \phi, \mathsf{T} \rangle$ such that the identifiers in T correspond to nodes of the form: $\langle \mathsf{id_p}, \gamma, \{\} \rangle$, where $\gamma \in \Gamma$ for all $\mathsf{id_p} \in \mathsf{T}$. This graph-based approach avoids the usual rigid row-based linearity of formal proofs (including those expressed in standard natural-deduction formats) in favor of more cognitively realistic "workspaces."

[3] For an overview of an earlier version of Slate, see [**?**].

3.2 META-R

At present, there are many excellent analogical reasoning systems in the literature, such as SME [**?**], LISA [**?**], and too many others to mention here. However, most if not all of these systems currently do not have the expressive power to represent arbitrarily complex multiply quantified formulas in first-order logic without contrived modifications. To meet the present challenge, such formulas must be represented in a manner that does not compromise the representation and reasoning used by Gödel (and other relevant thinkers). Furthermore, the definition of an optimal analogical match often differs from author to author. It seems to be universally agreed, nonetheless, that analogical matching involves some combination of heuristics, some of which may be highly domain-specific. These heuristics are often encoded into the matching algorithm itself. However, because we may be dealing with two domains that differ greatly in their level of formality, it is not clear what combination of heuristics would work best.

To answer this concern, the analogical matcher we chose to implement is *Modifiable Engine for Tree-based Analogical Reasoning* (META-R), based on the Flexible Heuristic Matcher (FHM) system described originally by Stephen Owen **?**. It is a blackboard-like system; such are known to perform well in unfamiliar domains where the search space may be rather large (as in tree matching, which we use here) and where there is a need to combine multiple lines of reasoning [**?**].

META-R takes as input two forests T_1, T_2 of labeled nodes, and a labeling function $\ell : (T_1 \cup T_2) \rightarrow LBL$ where LBL is the set of all unique labels in both domains. META-R outputs a node mapping $M \subseteq T_1 \times T_2$; and L, a mapping between the labels of T_1 and the labels of T_2. META-R has a set of matching rules (in blackboard terminology, these are *experts*) that are grouped into three levels, where rules on the same level have the same weight. META-R starts at the root nodes of the input forests and works its way down by asking the rules at the first level which descendants of the current nodes should be mapped to each other, giving precedence to the current nodes' children. If any unambiguous suggestions are made (meaning that more rules clearly suggest some pairing over another), those pairings are stored, and the children of the recently paired nodes are added to the 'to-do' queue. If a pair of nodes remains unpaired due to ambiguous suggestions, it is placed at the end of the queue and the decision as to how to pair them is deferred. When next those nodes are up for consideration, the next level of rules is consulted, and this process repeats until either all nodes are paired or some nodes exhaust all possible rules. The final level of rules consists of those we have created based on LP-rounding and bipartite matching (described shortly).[4] Where an assessment of match quality between two forests is needed, we adopt Owen's measure, which is simply a weighted combination of three percentages, each meant to reflect how well the match follows one of the three primary heuristics Owen identifies.

This is of course only a quick and incomplete summary of how META-R works, and we must keep it that way due to space; our intent is to express some of the strengths of META-R that make it a suitable choice for our present needs. Firstly, the analogical mapping problem differs from many tree- or graph-mapping problems, in that it does not only involve finding a node mapping, but a label mapping which likely is only partially available to the algorithm when it starts. Choices about which nodes are to be paired with each other in one part of the forest may affect decisions about which pairings are optimal to make in other parts of the forest, due to heuristics that attempt to ensure that nodes of certain labels map consistently across both domains. For example, consider the trees $T_1 = ((a \vee b) \wedge (b \vee a))$ and $T_2 = ((x \vee y) \wedge (x \vee y))$. Many naïve tree matching algorithms might pair $(a, x), (b, y)$ in one part, and $(b, x), (a, y)$ in the other (and the fact that choices made at one part of the tree or forest changes which choices are optimal in the other makes the solution space much more difficult to search than matching problems that start out with knowledge about optimal label matchings). In META-R, the tiered structure of the matching rules addresses this concern. The matching rules on the first level tend to check the current state of the mapping, in order to ensure consistency with decisions that have already been made. The later levels' rules tend to be more computationally intensive and able to operate independently of the current state of the mapping. In other words, if the algorithm is performing pairing at one part of the tree, and more information is needed before it can make a decision, it is often able to focus on another part of the tree first, where the best pairing choices may be more obvious.

Secondly, the choice of matching rules is not fixed by the algorithm. This gives us great flexibility to adopt some of the latest advances in matching algorithms without having to re-code the underlying program, two examples of which we have implemented and will summarize next. We start by defining two L-graphs L_1, L_2 for T_1, T_2 respectively, where an L-graph L_i simply consists of the original forest T_i plus one node for each unique label in the corresponding forest (these are called the 'label nodes'). For every node $n \in T_i$, an edge in L_i exists between n and its corresponding label node (note that this can destroy tree properties in the L-graph). We then define a linear program which attempts to find a subset of $(L_1 \cup T_1) \times (L_2 \cup T_2)$ which minimizes a linear combination of four penalties:

- Structural Consistency: How well the structures of the original forests are preserved. If two nodes are mapped to each other but their parents are not, a penalty of one is added. This allows an approximate adherence to the structure-mapping theory of analogy [**?**].
- Label Consistency: The linear program's constraints are defined such that label nodes can only be mapped to other label nodes, so that a label matching is chosen. A penalty is added for every node pairing which violates that label matching.
- One-to-one Consistency: A penalty is added for nodes that are in more than one pairing, and nodes that are in no pairings.
- Semantic and Pragmatic Constraints: Penalties can be used to encourage certain nodes or labels to be mapped together for reasons which may not reflected in the structure of the forests [**?**].

We then perform LP-rounding (using a threshold of 50%), and the resulting pairings are returned as suggestions. This

[4] Owen's original FHM split and examined all possible combinations of child pairs.

performs quite well on small input sizes (up to 50 nodes per side), but seems to break down on larger data sets.

In addition to the LP-rounding rule, we have a bipartite matching rule based on Riesen and Bunke's (2009) algorithm for approximate graph edit distance [**?**; **?**]. Graph edit distance is one of the most widely used measures of graph dissimilarity [**?**; **?**], so our matching rule operates on the assumption that a matching which minimizes the dissimilarity between two L-graphs corresponds to an optimal analogical match between the original forests. The algorithm essentially works by constructing a table that allows us to encode the same penalties as with our linear program, and applying the Kuhn-Munkres algorithm [**?**] to minimize the total penalty.

3.3 An Analogy to Reasoners: Smullyan's Island

It may be useful at this point to provide a high-level summary of the operations of the ADR system we have described:

1. The system accepts input; viz., a source domain S consisting of at least one full proof, domain-specific knowledge about the objects and predicates in that domain, and a target domain T consisting of a partial proof, domain-specific knowledge about the objects and predicates in that domain, and knowledge of some casting functions.
2. META-R performs analogical matching between S and T, and transfers a set of statements from source to target as S'.
3. S' is repaired using patching operators, and unfixable statements are dropped, to produce the set S''.
4. The statements in S'' are integrated into T in order, some statements becoming part of the partial proof in T.
5. If there is a goal statement in T, further deductive reasoning is applied to see if it follows from some combination of the new and old statements.

With this roadmap in place, we describe the particulars of the domains in question. We intend to demonstrate our system going from a semi-formal presentation of **LP** to a fully deductive proof of **G1**, using a semiformal version of **LP** described by Smullyan **?**. His version involves an island on which there exists a perfect reasoner. She can hold beliefs: written Bp for some proposition p. She can also reason about her beliefs; for example, the statement BBp means that she believes that she believes the proposition p. There is a very particular set of inference rules that she follows in order to produce new beliefs, of which she may not necessarily be aware (her reasoning process is automatic). For example, one such rule is $\forall_x(Bx \rightarrow BBx)$, which is not a valid FOL statement: if B is treated here as a predicate symbol then it can not be nested within itself. But if the logic is interpreted as semiformal and not entirely rigorous, then such details can be ignored as the intentions are clear.

Every day she produces exactly one new belief from her current set of beliefs and inference rules, such that every single belief she has and can possibly have is assigned a number corresponding to the day on which that belief was produced. This is denoted B_np, where n is the number corresponding to the day that the belief p was discovered. Smullyan then supposes that the following statements are true:

$\mathbf{C_{1a}}$ She believes all tautologies.
$\mathbf{C_{1b}}$ $\forall_{x,y}(Bx \wedge B(x \rightarrow y)) \rightarrow By$
$\mathbf{C_2}$ $\forall_{n,p}(B_np \rightarrow BB_np) \wedge (\neg B_np \rightarrow B\neg B_np)$
$\mathbf{C_3}$ $\forall_{n,p}B(B_np \rightarrow Bp)$
$\mathbf{S_{Con}}$ $\forall_p Bp \rightarrow \neg B\neg p$
$\mathbf{S_{\omega Con}}$ $\neg\exists_p(BBp \wedge \forall_n B\neg B_np)$
$\mathbf{D}$ $\forall_p Bp \leftrightarrow \exists_n B_np$

Smullyan then asks us to suppose that a native of the island comes to our ideal reasoner and declares: "You will never believe that I am a knight." A knight is someone who always tells the truth, as opposed to knaves, who only lie (the inhabitants of the island are all either knights or knaves). Since our ideal reasoner believes the rules of the island, she is led to believe that the native is a knight (k) if and only if what the native said ($\neg Bk$) is true:

$\mathbf{S}$ $B(k \leftrightarrow \neg Bk)$

From this and the premises listed above, Smullyan proves the goal statement $\mathbf{G}$:

$\mathbf{G}$ $\exists_p(\neg Bp \wedge \neg B\neg p)$

$\mathbf{G}$ can be informally understood as saying there exists a proposition that she will both never believe is true and never believe is false: for her, the statement is undecidable!

We ran our simulation using Smullyan's version of **LP** exactly as described above. **G1** was intentionally kept as sparse as possible, consisting only of meta-theoretic knowledge that one familiar with syntactic study of formal arithmetic theories might be expected to know. In the meta-language, the terms of the language stand for arithmetic theories, formulae and numerals standing for object-level proofs, terms, and formulae. All the syntactic objects used in Figure 1 and Figure 2 belong to the meta-language. In both the figures, the constant symbol Φ stands for the particular first-order arithmetic theory we are interested in, namely Peano Arithmetic. The following statements were included:

$$\mathbf{D'}\ \forall_q(Proves(\Phi, q) \leftrightarrow \exists_n Prf(n, numeral(q)))$$

$$\mathbf{S'_{Con}}\ \forall_q Proves(\Phi, q) \rightarrow \neg Proves(\Phi, \neg q)$$

$$\mathbf{S'_{\omega Con}}\ \begin{array}{l}\neg\exists_q(Proves(\Phi, apply(\rho, numeral(q))) \\ \wedge\forall_n Proves(\Phi, not(prf(n, numeral(q)))))\end{array}$$

$$\mathbf{G'}\ \exists_\phi(\neg Proves(\Phi, \phi) \wedge \neg Proves(\Phi, not(\phi)))$$

The above statements constitute a partial first-order axiomatization of the metatheory. Predicate symbols are capitalized, and function symbols are lower-cased. $Proves(\Phi, p)$ holds when p is a theorem of Φ. If q is a well-formed formula and $numeral(q)$ is its Gödel numeral, $apply(\rho, numeral(q))$ stands for formula with the unary property ρ applied to the Gödel numeral of the formula q (this might also be written: $\rho(\ulcorner q \urcorner)$). $Prf(n, m)$, borrowed from [**?**] is a meta-theoretic relation symbol shadowing the object-theoretic $\mathsf{Prf(n, m)}$ from Smith. $\mathsf{Prf(n, m)}$ holds just when n is the numeral for the Peano-Arithmetic proof of the statement for which m is the numeral. The term representing $\mathsf{Prf(n, m)}$ in the metalanguage is $prf(n, m)$.

3.4 Results

META-R correctly matched $\mathbf{S_{Con}}$, $\mathbf{S_{\omega Con}}$, $\mathbf{D}$ and $\mathbf{G}$ with $\mathbf{S'_{Con}}$, $\mathbf{S'_{\omega Con}}$, $\mathbf{D'}$ and $\mathbf{G'}$, respectively. From these, it matched the B_n operator with the Prf predicate and its function version prf, and the B operator with both the $Proves$

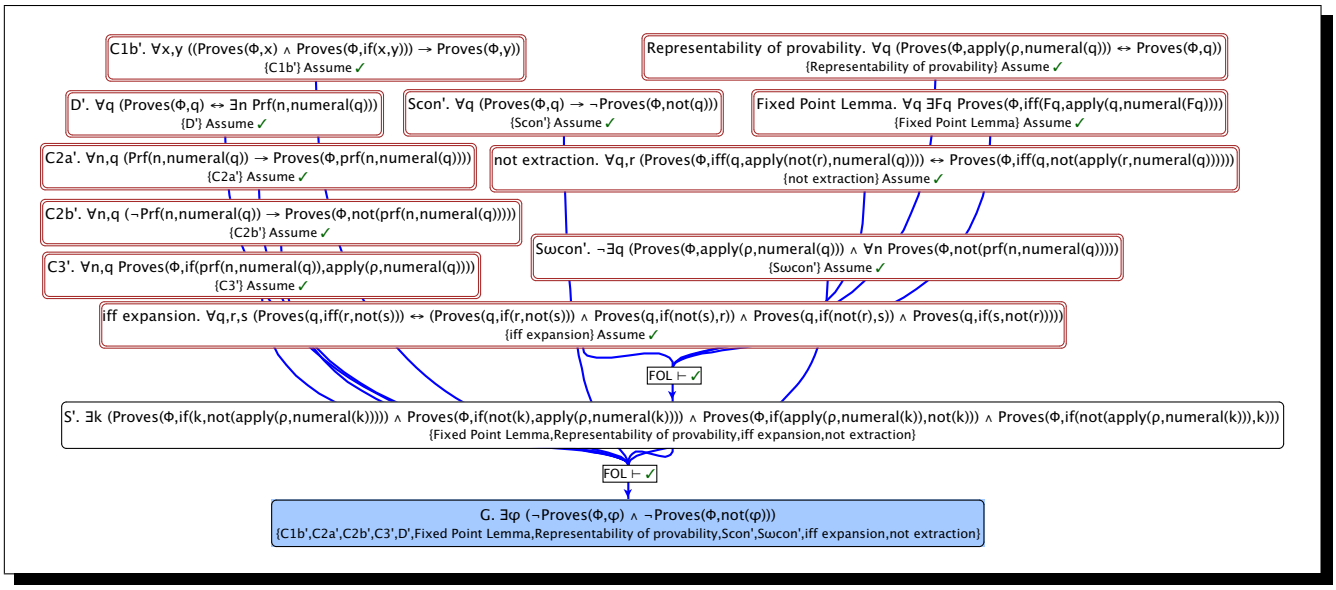

Figure 2: A full proof of **G1** in Slate, generated through ADR. Note that $\mathbf{S}'$ is an expanded version of what actually follows from the analogical transfer of **S**; this was done for better presentability of this image.

predicate and the *apply* function. Because no mechanism was in place to allow the system to understand that one was to be used only as a predicate and the other as a function symbol, during the translation step formulas such as $\forall_{n,q} Proves(\Phi, prf(n, numeral(q)) \rightarrow Proves(\Phi, q))$ were generated. However, because no patching operator exists that could cast that statement into a well-formed one, it was discarded.

The resulting proof, tidied up and presented in Slate, is pictured in Figure 2. Note that two boxes are taken from the manually constructed proof in Figure 1 (Fixed Point Lemma and Representability of Provability, though the latter is not pictured in Figure 1). There are also two boxes included in order to allow for trivial syntactic transformations. The resulting proof is comparable to the one in Figure 1, though clearly missing some of the intermediate steps.

4 Conclusion

Analogy-assisted automated theorem proving is a rich topic about which much has been published; for economy we just mention a few directly relevant papers here. [**?**] is an early paper discussing the potential of integrating analogical reasoning and first-principles reasoning. Our method of cross-domain knowledge transfer is similar to that in [**?**]. Our paper might be considered an implementation of their work, applied to the realm of **G1**. Melis **?** is another major contributor to this field. Her system involves analogy and proof-planning through the use of operators known as *methods*. Because the present application of analogical matching did not involve the analogical transfer of all intermediate steps in a proof as large as Figure 1, we did not run into many of the problems that Melis' work is designed to circumvent; however, in future work such techniques will likely need to be adopted.

4.1 Future Work

Though the present work has attempted to bridge the gap between analogical and deductive reasoning in a way that resembles human-level thought, we are careful not to overstate our claims. Much more work is needed before the process which we have automated here, designed for the domains of **LP** and **G1**, can be effortlessly applied to other problems. We will next attempt to apply the lessons learned here to the well-known Tarski's Theorem, which invokes similar concepts and for which Smullyan's analogies may also be useful.

Regarding META-R development, further testing is needed to determine to what degree the use of our LP-rounding and bipartite matching rules are sufficient alternatives to Owen's original approach, which branches through all possible combinations. In our experience, however, the difference has not been noticeable. As far as we know, ours is the only existing FHM-based implementation currently in use, and we hope to continue its development and make it available to researchers interested in experimenting with analogical matching.

Finally, we will explore another method of proving **G1** that we suspect will produce interesting results. This has been aptly called the "Master Argument" by Smith **?**; it actually matches the route that Gödel himself regarded to be the most perspicuous one to incompleteness, and uses Tarski's Theorem, which in point of fact Gödel proved before Tarski.