A Program to Compute Gödel-Löb Fixpoints

Melvin Fitting
mlflc@cunyvm.cuny.edu
Dept. Mathematics and Computer Science
Lehman College (CUNY), Bronx, NY 10468
Depts. Computer Science, Philosophy, Mathematics
Graduate Center (CUNY), 33 West 42nd Street, NYC, NY 10036

October 4, 1995

1 Introduction

Briefly, the purpose of this paper is to describe a Prolog program that computes fixpoints in Gödel-Löb computability logic (*GL*). In order to make things relatively self-contained, I sketch the essential ideas of *GL*, and discuss the significance of its fixpoint theorem. Then I give the algorithm embodied in the program in a little more detail. It should be emphasized that nothing new is presented here — all the theory and methodology are due to others. The main interest is, in a sense, psychological. The approach taken here has been declared in the literature, more than once, to be of theoretical interest only, being too unwieldly in practice. Experimentation shows this is not so provided formulas are not too complicated. A copy of the program can be obtained by ftp from venus.gc.cuny.edu. It is in the subdirectory "/pub/fitting," under the name "glfixpt.pro." Log on as "anonymous" and use your full e-mail address as password. If there are difficulties with ftp, send e-mail to the author at mlflc@cunyvm.cuny.edu. The Prolog code is standard, and should run under any implementation, though minor modifications may be necessary for some systems. These modifications are described in the last section, and also in the program itself.

2 Gödel-Löb logic

Gödel originated the idea of treating provability as a modal notion [6]. The technique is as follows.

On the one hand we have a propositional modal language, in which formulas are built up from propositional variables, P, Q, …, and propositional constants, $\top$ (truth), and $\bot$ (falsehood), using $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$, $\Box$ and $\Diamond$ in the usual way. On the other hand we have Peano arithmetic, *PA*, which I assume does not need a description.

Let θ be an arbitrary mapping from the propositional variables of the modal language to closed formulas of *PA*. The action of θ is extended to all modal formulas as follows.

$$\begin{aligned}\theta(\bot) &= (0 = 1)\\ \theta(\top) &= (0 = 0)\\ \theta(X \supset Y) &= \theta(X) \supset \theta(Y)\end{aligned}$$

$$\theta(\neg X) \quad = \quad \neg\theta(X)$$

and the most significant item

$$\theta(\Box X) \quad = \quad \mathit{Prv}(\text{“}\theta(X)\text{”})$$

In this, *Prv* is Gödel's provability predicate, and “Z” is the Gödel number of the sentence Z. Thus the θ-translation of $\Box X$ is the assertion, within *PA*, that the translation of X is provable. Incidentally, translations of connectives and modalities not explicitly mentioned above are treated, via the usual definitions, in terms of those that are.

Now, call a modal formula X *valid* if, for every θ, $\theta(X)$ is a theorem of *PA*. This defines a modal logic, often called *provabiliy logic*. Gödel himself observed, of this logic, that all tautologies are valid, that the validities are closed under modus ponens and the rule of necessitation (conclude $\Box X$ from X, a rule due to Gödel), and that all instances of $\Box(X \supset Y) \supset (\Box X \supset \Box Y)$ and $\Box X \supset \Box\Box X$ are valid. He also noted that not all instances of $\Box X \supset X$ are valid. In particular, $\Box\bot \supset \bot$, or equivalently, $\neg\Box\bot$, always translates to $\neg \mathit{Prv}(\text{“}0 = 1\text{”})$. If this were provable in *PA*, then *PA* would prove its own consistency, and Gödel had showed it could not (if consistent). In modern terminology, provability logic is somewhere between *K4* and *S4*, but a more precise characterization was left open.

In proving incompleteness, Gödel made use of a sentence of arithmetic that asserted its own unprovability. Henkin raised the question of the status of a similar sentence that asserted its own provability. Here the outcome is not at all obvious. Löb answered the question with a remarkable result [7], now known as Löb's theorem: If A is a sentence of *PA* for which $\mathit{Prv}(\text{“}A\text{”}) \supset A$ is provable in *PA*, then A itself is a theorem. Converting Löb's result to the modal setting, it becomes the rule:

$$\frac{\Box X \supset X}{X}$$

Adding this rule to the modal principles Gödel mentioned (that is, to *K4*) yields a logic now known as *GL*, for Gödel and Löb. More precisely, *GL* is the modal logic characterized as follows.

Axiom Schemas

$$\begin{array}{l} \text{All tautologies} \\ \Box(X \supset Y) \supset (\Box X \supset \Box Y) \\ \Box X \supset \Box\Box X \end{array}$$

Modus Ponens

$$\frac{X \qquad X \supset Y}{Y}$$

Necessitation

$$\frac{X}{\Box X}$$

Löb's Rule

$$\frac{\Box X \supset X}{X}$$

In general, adding new rules of inference is felt to be less desirable than adding new axioms. Fortunately Löb's rule can be replaced by an axiom schema, to give an equivalent system.

Löb's Axiom $\Box(\Box X \supset X) \supset \Box X$

Here is a sketch of a proof of the equivalence of the two formulations. Suppose we have the axiom scheme as primitive, and $\Box X \supset X$ has been proved. By the Necessitation Rule we have $\Box(\Box X \supset X)$. Then by Löb's Axiom, and Modus Ponens, we have $\Box X$. Finally, since we have $\Box X \supset X$, by Modus Ponens again, we get X. The other direction is more work. Suppose we have the rule as primitive. Let Z be the modal formula $\Box(\Box X \supset X) \supset \Box X$, which we wish to prove. Löb's Rule says we have a proof of it if we can prove $\Box Z \supset Z$. In fact this can be shown in *K4*, which is the system above without either Löb's Axiom or Löb's Rule. This I leave to you.

Customarily, the modal logic *GL* is formulated axiomatically using Löb's Axiom, rather than the Rule. Between the work of Gödel and Löb, every modal formula provable in *GL* is valid — all translations into arithmetic yield sentences provable in *PA*. Solovay proved the remarkable result that the converse is also true [11], and thus *GL* is exactly the modal counterpart of *PA* provability. Solovay's proof is much too complex to be even sketched here. Suffice it to say that it involves a difficult fixpoint construction, using either the second recursion theorem of recursion theory, or Gödel's fixedpoint result for *PA*. Any of [1, 2, 9] provide thorough treatments of *GL*.

3 The Fixed Point Theorem

Gödel's proof of his incompleteness theorems depended on finding a sentence of arithmetic that expressed its own unprovability. More generally, he developed a technique for constructing "fixpoint" sentences that express facts about themselves — a technique involving diagonalization. I remarked above that this is used in the proof of Solovay's completeness theorem for *GL*. Speaking informally, in modal terms Gödel produced a sentence P such that $\neg\Box P \equiv P$. One of the triumphs of the modal approach to arithmetic provability is the de Jongh, Sambin result that says, roughly: in *GL* fixed points always exist. That is, the modal system is powerful enough to capture every application of Gödel's diagonal argument — and it does so in a *propositional* setting!

First some terminololgy and notation. Let $A(P)$ be a propositional modal formula, where P is a propositional variable. By $A(X)$ is meant the result of replacing all occurrences of P in $A(P)$ with occurrences of the modal formula X. A formula is *modalized in* P if every occurrence of P in it is within the scope of a modal operator.

Fixed Point Theorem If $A(P)$ is modalized in P, there is a fixed point formula D such that $A(D) \equiv D$ is a theorem of *GL*. Further, D is unique in the sense that if $A(D) \equiv D$ and $A(D') \equiv D'$ are both provable in *GL*, then so is $D \equiv D'$.

There are several proofs of this, generally constructive in nature. The most elegant derives the existence of fixed points from the Craig interpolation theorem for *GL*, [8], though the authors remark, "But the calculations involved, even in simple cases, are too complex to be carried out in practice. Thus a faster method of finding the fixed point would be useful." In fact, they do proceed to give another constructive method. But we will see that the method based on interpolation is easy to program and reasonably efficient in practice.

In the remainder of this section I briefly sketch the derivation of the existence of fixed points using the interpolation theorem, following the argument in [8]. To do this we need a stronger form of the uniqueness of fixed points than was given above. Let $\triangle X$ abbreviate $\Box X \wedge X$.

Fixed Point Uniqueness Suppose $A(P)$ is modalized in P, and Q is a new propositional variable. The following is a theorem of *GL*.

$$[\triangle(A(P) \equiv P) \wedge \triangle(A(Q) \equiv Q)] \supset (P \equiv Q).$$

Note that the version of uniqueness stated in the Fixed Point Theorem above follows easily, using Necessitation and Modus Ponens.

The key item needed to prove Fixed Point Uniqueness is this: if $A(P)$ is modalized in P, and Q is new, then $\Box(P \equiv Q) \supset [A(P) \equiv A(Q)]$ is a theorem of *GL*. This is proved rather easily using the tableau formulation of the next section — the proof is omitted here. Now, using this, the following is obviously a theorem:

$$[\Box(P \equiv Q) \wedge (A(P) \equiv P) \wedge (A(Q) \equiv Q)] \supset (P \equiv Q)$$

Then also

$$[(A(P) \equiv P) \wedge (A(Q) \equiv Q)] \supset [\Box(P \equiv Q) \supset (P \equiv Q)]$$

so by the Necessitation Rule, and the distributivity of $\Box$ over implication and conjunction,

$$[\Box(A(P) \equiv P) \wedge \Box(A(Q) \equiv Q)] \supset \Box[\Box(P \equiv Q) \supset (P \equiv Q)]$$

But by Löb's Axiom $\Box[\Box(P \equiv Q) \supset (P \equiv Q)] \supset \Box(P \equiv Q)$, and we also know $\Box(P \equiv Q) \supset (A(P) \equiv A(Q))$, so

$$[\Box(A(P) \equiv P) \wedge \Box(A(Q) \equiv Q)] \supset [A(P) \equiv A(Q)]$$

from which it trivially follows that

$$[\triangle(A(P) \equiv P) \wedge \triangle(A(Q) \equiv Q)] \supset (P \equiv Q).$$

In addition we need one more technical fact, which also follows easily from the tableau formulation of *GL* to be given in the next section.

Technical Fact Suppose the propositional variable P is modalized in both X and Y. If $(P \wedge X) \supset Y$ is provable in *GL*, so is $X \supset Y$. Similarly if $X \supset (P \vee Y)$ is provable, so is $X \supset Y$.

Now we show the existence of fixed points, using the proof from [8]. Assume $A(P)$ is modalized in P, and Q is a new propositional variable. The following is a theorem of *GL*:

$$[P \wedge A(P) \wedge \Box(A(P) \equiv P)] \supset \{[(A(Q) \equiv Q) \wedge \Box(A(Q) \equiv Q)] \supset (A(Q) \vee Q)\}$$

Reasons: $P \wedge A(P) \wedge \Box(A(P) \equiv P)$ implies $\triangle(A(P) \equiv P)$; $(A(Q) \equiv Q) \wedge \Box(A(Q) \equiv Q)$ is $\triangle(A(Q) \equiv Q)$; we have the strong form of fixed point uniqueness; hence from these follow $P \supset Q$. Since the antecedent contains P as a conjunct and the consequent contains Q as a disjunct, the formula is provable.

The following is also a theorem of *GL*:

$$[P \wedge A(P) \wedge \Box(A(P) \equiv P)] \supset \{[\neg(A(Q) \equiv Q) \wedge \Box(A(Q) \equiv Q)] \supset (A(Q) \vee Q)\}$$

This is so simply because $\neg(A(Q) \equiv Q)$ classically implies $A(Q) \vee Q$. Using these two items, the following (which is what we really want) is a theorem of *GL*:

$$[P \wedge A(P) \wedge \Box(A(P) \equiv P)] \supset [\Box(A(Q) \equiv Q) \supset (A(Q) \vee Q)]$$

Then by the Craig theorem (which will be proved below for *GL*) there is an interpolant D for this, so that $[P \wedge A(P) \wedge \Box(A(P) \equiv P)] \supset D$ and $D \supset [\Box(A(Q) \equiv Q) \supset (A(Q) \vee Q)]$ are *GL* theorems, and all propositional variables of D are common to both $P \wedge A(P) \wedge \Box(A(P) \equiv P)$ and

$\Box(A(Q) \equiv Q) \supset (A(Q) \vee Q)$. It follows that neither P nor Q can occur in D (in particular, P is vacuously modalized in D).

Since $[P \wedge A(P) \wedge \Box(A(P) \equiv P)] \supset D$ is a theorem, by the Technical Fact, so is $[A(P) \wedge \Box(A(P) \equiv P)] \supset D$, and hence also $\Box(A(P) \equiv P) \supset (A(P) \supset D)$.

Since $D \supset [\Box(A(Q) \equiv Q) \supset (A(Q) \vee Q)]$ is a theorem of *GL*, so is the result of replacing all Q's with P's, and since Q does not occur in D, this gives $D \supset [\Box(A(P) \equiv P) \supset (A(P) \vee P)]$. Using the Technical Fact again, $D \supset [\Box(A(P) \equiv P) \supset A(P)]$ is a theorem, hence also $\Box(A(P) \equiv P) \supset (D \supset A(P))$.

Now, $\Box(A(P) \equiv P) \supset (A(P) \equiv D)$ is a theorem. Then so is the result of uniformly substituting an arbitrary formula for P — substitute D itself. Then $\Box(A(D) \equiv D) \supset (A(D) \equiv D)$ is a theorem of *GL*. Finally, by Löb's Rule, $A(D) \equiv D$ is a theorem, so D is a fixed point for $A(P)$.

Thus the construction of a fixed point for $A(P)$ is, at least in principle, simple: it is an interpolant for

$$[P \wedge A(P) \wedge \Box(A(P) \equiv P)] \supset [\Box(A(Q) \equiv Q) \supset (A(Q) \vee Q)].$$

4 A Tableau System

To compute fixed points, we need a constructive proof of the Craig theorem for *GL*, and that is easily given once a tableau (or Gentzen) system is available. I use a formulation from [3] — the idea is the same as in [1]. Since it makes the programming somewhat easier, I use *uniform notation.* I begin with the well-known propositional system of Smullyan [10] (see also [5]).

A category of so-called α or conjunctive formulas is defined in the following table and, for each, two components, α_1 and α_2, are defined as well.

α	α_1	α_2
$X \wedge Y$	X	Y
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \supset Y)$	X	$\neg Y$

Likewise β or disjunctive formulas, and their components are defined as follows.

β	β_1	β_2
$X \vee Y$	X	Y
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$X \supset Y$	$\neg X$	Y

(Incidentally, we can also bring $\equiv$ into the picture by treating $X \equiv Y$ as α, with $\alpha_1 = (X \supset Y)$ and $\alpha_2 = (Y \supset X)$, and similarly for $\neg(X \equiv Y)$. The Prolog program takes this approach.)

Now, a classical tableau proof of a propositional formula X is a closed tableau for $\neg X$. A tableau for $\neg X$ is a tree beginning with $\neg X$ at the root, and "grown" using the following *branch extension rules.*

$$\frac{\neg\neg Z}{Z} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top} \qquad \frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2}$$

A tableau branch is closed if it contains $\bot$ or if it contains Z and $\neg Z$ for some Z. A tableau is closed if every branch is closed.

The classical rules above are *non-destructive*, branches grow from their endpoints, and nothing disappears or changes. The system can be extended to modal logic by adding certain *destructive* rules that change branches considerably. For clarity I begin with a system for K, the smallest normal modal logic. A fuller discussion can be found in [3, 4]. First, two new categories of formulas are defined, necessary or ν formulas, and possible or π formulas, and for each, one component.

ν	ν_0	π	π_0
$\Box X$	X	$\Diamond X$	X
$\neg\Diamond X$	$\neg X$	$\neg\Box Y$	$\neg X$

Also, for a set S of formulas, define a set $S^{\#}$ as follows:

$$S^{\#} = \{\nu_0 \mid \nu \in S\}.$$

Now the single additional modal rule is the following:

$$\frac{S \cup \{\pi\}}{S^{\#} \cup \{\pi_0\}}$$

The intended use of the rule is this: if $S \cup \{\pi\}$ is the set of formulas on a branch, *the entire branch may be replaced with* $S^{\#} \cup \{\pi_0\}$. Notice that since this rule destroys or modifies formulas, and can often be applied in more than one way on a branch (since several formulas of type π may be on a branch) backtracking becomes necessary. This is not the case classically, and accounts for the added complexity of most modal logics.

The system just described is sound and complete with respect to K. To add transitivity, getting $K4$, we simply redefine the # operator.

$$S^{\#} = \{\nu, \nu_0 \mid \nu \in S\}.$$

And finally, to get GL we use the redefined # operator, and change the corresponding rule to the following.

$$\frac{S \cup \{\pi\}}{S^{\#} \cup \{\pi_0, \neg\pi\}}$$

To show the resulting tableau system is equivalent to the axiomatic version given earlier, it is easiest to show both are sound and complete with respect to an appropriate Kripke semantics. There are several ways of doing this. A uniform approach using *consistency properties* can be found in [3].

5 Interpolation

Smullyan gave a constructive proof of the classical Craig theorem using what he called a *symmetric Gentzen system*. In [3] this was extended to prove interpolation for GL, among other modal logics. What follows is a variation on that proof.

To prove $X \supset Y$ using tableaus we begin with $\neg(X \supset Y)$, add X and $\neg Y$, and go on from there to a closed tableau. It can be shown that $\neg(X \supset Y)$ will never be needed again, so we may as well just start with X and $\neg Y$. Then each subsequent addition is either an descendant of X or of $\neg Y$, in an obvious way. The idea is to keep track of this descendancy, then use it and the closed tableau that we constructed, to build an interpolant for $X \supset Y$. To this end we introduce *biased formulas*, and modify tableaus so they are used instead of ordinary ones. Biased formulas are expressions of the form $L(Z)$ or $R(Z)$, where L and R stand for "left" and "right" respectively. The intention is that if $L(Z)$ appears in a tableau, Z is a descendant of X, and if $R(Z)$ appears, Z is a descendant of $\neg Y$, corresponding to the left and right sides of $X \supset Y$.

We modify the uniform notation system of the previous section, so that it applies to biased formulas. The modification is straigtforward. For instance, if Z is an α, with components α_1 and α_2, we say $L(Z)$ is also an α, and its components are $L(\alpha_1)$ and $L(\alpha_2)$. Similarly for all the other cases. Then, formally, the tableau rules are exactly as before.

Finally, a tableau branch involving biased formulas is closed if it contains a syntactic contradiction *ignoring the signs*. Thus a branch containing $L(A)$ and $R(\neg A)$ is closed, as is one containing $L(A)$ and $L(\neg A)$, and so on. Note that a closed biased tableau beginning with $L(X)$ and $R(\neg Y)$ becomes an ordinary closed tableau if "L" and "R" occurrences are erased. Also a closed ordinary tableau beginning with X and $\neg Y$ can easily be turned into a closed biased tableau by systematically inserting "L" and "R" occurrences in the obvious way. Thus we can assume that $X \supset Y$ is a theorem of *GL* if and only if there is a closed biased tableau beginning with $L(X)$ and $R(\neg Y)$.

By an *interpolant* for a set of biased formulas $\{L(X_1), \ldots, L(X_n), R(Y_1), \ldots, R(Y_k)\}$ we mean a formula Z such that

1. $(X_1 \wedge \cdots \wedge X_n) \supset Z$ is a theorem of *GL*,
2. $Z \supset (\neg Y_1 \vee \cdots \vee \neg Y_k)$ is a theorem of *GL*,
3. all propositional variables of Z are common to both $\{X_1, \ldots, X_n\}$ and $\{Y_1, \ldots, Y_k\}$.

We say a biased tableau branch has an interpolant if the set of biased formulas on it does. Suppose there is a closed biased tableau beginning with $L(X)$ and $R(\neg Y)$. We show there is an interpolant for each closed branch and then, successively, there are interpolants that result from the "undoing" of each branch extension rule application, one by one. Then finally we have an interpolant for the original tableau, before any rules were applied, the tableau containing only $L(X)$ and $R(\neg Y)$, and it is easy to see that an interpolant for the set consisting of these two is an interpolant, in the usual sense, for $X \supset Y$.

I use the following notation: $S \xrightarrow{int} Z$ to symbolize that Z is an interpolant for the finite set S of biased sentences. For simplicity, I do not allow $\top$ or $\bot$ in S (though they may occur as parts of Z). Now, first, here are the conditions for closed branches.

$$\begin{array}{rcl} S \cup \{L(A), L(\neg A)\} & \xrightarrow{int} & \bot \\ S \cup \{R(A), R(\neg A)\} & \xrightarrow{int} & \top \\ S \cup \{L(A), R(\neg A)\} & \xrightarrow{int} & A \\ S \cup \{R(A), L(\neg A)\} & \xrightarrow{int} & \neg A \end{array}$$

Next the easy propositional cases.

$$\frac{S \cup \{L(Z)\} \xrightarrow{int} A}{S \cup \{L(\neg\neg Z)\} \xrightarrow{int} A} \qquad \frac{S \cup \{R(Z)\} \xrightarrow{int} A}{S \cup \{R(\neg\neg Z)\} \xrightarrow{int} A}$$

The α cases are also straightforward.

$$\frac{S \cup \{L(\alpha_1), L(\alpha_2)\} \xrightarrow{int} A}{S \cup \{L(\alpha)\} \xrightarrow{int} A} \qquad \frac{S \cup \{R(\alpha_1), R(\alpha_2)\} \xrightarrow{int} A}{S \cup \{R(\alpha)\} \xrightarrow{int} A}$$

Finally the β cases, which are the most interesting of the classical rules.

$$\frac{S \cup \{L(\beta_1)\} \xrightarrow{int} A \quad S \cup \{L(\beta_2)\} \xrightarrow{int} B}{S \cup \{L(\beta)\} \xrightarrow{int} A \vee B} \qquad \frac{S \cup \{R(\beta_1)\} \xrightarrow{int} A \quad S \cup \{R(\beta_2)\} \xrightarrow{int} B}{S \cup \{R(\beta)\} \xrightarrow{int} A \wedge B}$$

And, to finish things, the modal rules.

$$\frac{S^{\#} \cup \{L(\pi_0), L(\neg\pi)\} \xrightarrow{int} A}{S \cup \{L(\pi)\} \xrightarrow{int} \Diamond A} \qquad \frac{S^{\#} \cup \{R(\pi_0), R(\neg\pi)\} \xrightarrow{int} A}{S \cup \{R(\pi)\} \xrightarrow{int} \Box A}$$

I leave it to you to carry out the straightforward checking that these rules produce appropriate interpolants.

6 Implementation

As mentioned in the Introduction, a Prolog implementation of what has been presented here is available by anonymous ftp. The implementation was designed for readability, not for speed or compactness. I sketch the basic ideas of the implementation, and conclude with instructions for using it.

First, Prolog makes it easy to define infix and prefix operators. Taking advantage of this, I treat "and," "or," "imp," and "iff" as infix, and "neg," "box," and "dia" as prefix. (These have their obvious meanings.) This is done using Prolog's `op` command. For example, `?-op(120, yfx, or).` sets `or` to be infix, have a precedence level of 120, and be left associative. (Parentheses can still be used for clarity.) A word of caution here, however. Some Prologs want these instructions to begin with `?-`, some with `:-`, and a few with nothing. Check your system, and make corresponding adjustments.

There is one more modification you may need to make, depending on your system. Some Prologs have a built-in `copy_term` predicate, and some do not. If yours does not, it can be programmed using 'standard' Prolog; the necessary material will be found at the start of the program. No other changes should be needed to adapt the program to various versions of Prolog.

The program has a few standard utility predicates, for membership in a list, and for removal of items from lists. After these, the implementation properly begins.

Conjunctive formulas are defined by simply enumerating all possible cases. For example, both `(p and q)` and `neg(p or q)` are conjunctive. Similarly, disjunctive formulas are defined, as are negative ones (involving double negations), necessities, and possibilities. In addition, biased versions of these are defined, by reducing them to the unbiased versions. For example, `left(p and q)` is conjunctive because `(p and q)` is. (We use `left` as the Prolog counterpart of L from section 5.)

The next part of the program defines, for each conjunctive, or α-formula, its components, and similarly for disjunctive, or β-formulas. Likewise for formulas involving unary connectives or modalities, the component is defined. Again, this is done by enumerating all cases.

Think of a tableau branch as a *list* of biased formulas. For such a list L, the sharp operation, resulting in $L^{\#}$, is defined next. This is straightforward.

Now comes the heart of the program — the part that calculates interpolation formulas. It consists of a number of clauses, each representing one case. Two will suffice for illustration. First, a branch closure case.

```
interp(Br, I) :-
   member(left(A), Br),
   member(left(neg A), Br),
   I = false.
```

Informally, this says a branch `Br` of a biased tableau that contains the two biased formulas `left(A)` and `left(neg A)` has `false` as interpolant. Next, a more elaborate reduction case.

```
interp(Br, I) :-
   member(Beta, Br),
   Beta = left(_),
   disjunctive(Beta),
   components(Beta, Beta_one, Beta_two),
   remove(Beta, Br, Temp),
   New_Br_one = [Beta_one | Temp],
   New_Br_two = [Beta_two | Temp],
   interp(New_Br_one, A),
   interp(New_Br_two, B),
   I = (A or B).
```

This says that `(A or B)` is an interpolant for a biased tableau branch `Br` provided: there is a biased formula `Beta` on the branch that is disjunctive and of the form `left(something)`; its components are `Beta_one` and `Beta_two`; `A` is an interpolant for a branch like `Br` but with `Beta_one` in place of `Beta`; and `B` is an interpolant for a branch like `Br` but with `Beta_two` in place of `Beta`. The connection between this and the corresponding rule in section 5 should be obvious.

Interpolants calculated in this way may be unnecessarily complicated. The next part of the program applies some elementary simplifications. For example, `(true and X)` is replaced with `X`.

Finally, we reach the program driver. Say we want a fixed point for the formula $\varphi(P)$, where P is the fixpoint variable, and there may be other atomic formulas besides P. Choose a Prolog variable, say `P`, for the fixpoint variable, and use Prolog constants for the other atomic formulas. Rewrite the formula in Prolog notation (we give an explicit example below). Say the rewritten formula is `F(P)`. Now issue the Prolog query: `fixedpoint(F(P), P)`, where we specify not only the formula, but the fixpoint variable as well.

What we want is an interpolant for the formula

$$[p \wedge \varphi(p) \wedge \Box(\varphi(p) \equiv p)] \supset [\Box(\varphi(q) \equiv q) \supset (\varphi(q) \vee q)]$$

where p and q are new propositional variables. The call on `fixedpoint` accomplishes this in the following way. Using `copy_term`, the fixpoint variable `P` is replaced in `F(P)` with the arbitrary constant `ppp`, to produce a formula called `AP`; then again, `P` is replaced in `F(P)` with the constant `qqq`, to produce a formula called `AQ`. (We use `ppp` instead of `p` because it is unlikely a user would pick such a variable to use in a formula, so it should be new. Similarly for `qqq`.) Next, formulas corresponding to the left and right sides of the implication above are formed (they are called `One` and `Two` in the program), and the interpolant calculator, `interp`, is called on. Its result is simplified, and printed.

For a concrete example, suppose we want an interpolant for the formula $\Box P \supset (Q \supset \Box\neg P)$, where P is the fixpoint variable, and Q is thought of as a parameter. We choose a Prolog variable, say `P`, for the fixpoint variable, and a Prolog constant, say `q`, for Q, so the formula, in Prolog notation, is `box P imp (q imp box neg P)`. Now we issue the Prolog query:

```
fixedpoint(box P imp (q imp box neg P), P)
```

where we specify the formula and the fixpoint variable. Prolog returns the answer:

```
dia (q and dia true) or (neg q or box false)
```

which in conventional notation is $\Diamond(Q \wedge \Diamond\top) \vee (\neg Q \vee \Box\bot)$. Likewise, a fixpoint for $\Box P \supset \Box\neg P$ will easily be determined: $\Diamond\Diamond\top \vee \Box\bot$ (consistency is consistent, or else inconsistency). Notice that this is a special case of the previous example, taking $Q = \top$. I leave it to you to verify that these really are fixpoints.

I conclude by repeating what I said earlier — there is nothing original in this paper or in the program. What is truly astounding, however, is the cumulative work of the many researchers that led us to this point — where Gödel's insights have been turned into mechanical calculations that can still impress and inspire.

References

[1] Boolos, G. *The Unprovability of Consistency.* Cambridge University Press, 1979.

[2] Boolos, G. *The Logic of Provability.* Cambridge University Press, Cambridge, New York, 1993.

[3] Fitting, M. C. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishing Co., Dordrecht, 1983.

[4] Fitting, M. C. First-order modal tableaux. *Journal of Automated Reasoning 4* (1988), 191–213.

[5] Fitting, M. C. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1990.

[6] Gödel, K. An interpretation of the intuitionistic propositional calculus. In *Kurt Gödel, Collected Works, Volume One*, S. Feferman, Ed. Oxford, 1986, pp. 300–303. Originally published as 'Eine Interpretation des intuitionistischen Aussagenkalküls', in *Ergebnisse eines mathematischen Kolloquiums*, vol 4, pp 39–40 (1933).

[7] Löb, M. H. Solution of a problem of Leon Henkin. *Journal of Symbolic Logic 20* (1955), 115–118.

[8] Sambin, G., and Valentini, S. The modal logic of provability, the sequential approach. *Journal of Philosophical Logic 11*, 3 (1982), 311–342.

[9] Smoryński, C. *Self-reference and Modal Logic.* Universitext, Springer-Verlag, 1985.

[10] Smullyan, R. M. *First-Order Logic.* Springer-Verlag, 1968. Somewhat revised edition, Dover Press, NY, 1994.

[11] Solovay, R. M. Provability interpretations of modal logic. *Israel Journal of Mathematics 25* (1976), 287–304.