Porgi: a Proof-Or-Refutation Generator for Intuitionistic propositional logic[*]

Allen Stoughton[†]
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS 66506, USA
`allen@cis.ksu.edu`
`http://www.cis.ksu.edu/~allen/home.html`

Abstract. Porgi is a Proof-Or-Refutation Generator for Intuitionistic propositional logic. Given a sequent, Porgi either finds a minimally sized, normal natural deduction of the sequent, or it finds a "small", tree-based Kripke countermodel of the sequent. Porgi is implemented in Standard ML, and can be obtained via WWW URL `http://www.cis.ksu.edu/~allen/porgi.html`.

1 Introduction

Porgi is a Proof-Or-Refutation Generator for Intuitionistic propositional logic. Given a sequent $\Gamma \Rightarrow \varphi$, Porgi either finds a minimally sized, normal natural deduction of φ from the assumptions in Γ, or it finds a finite, tree-based Kripke model whose root node forces all of the formulas in Γ but does not force φ. (The *size* of a natural deduction (respectively, Kripke model) is the number of nodes in the deduction (respectively, model).) Although an attempt is made to minimize the size of the Kripke countermodels, such countermodels are not always minimally sized. On the other hand:

(a) Classical models are produced whenever possible. Thus, if a model with more than one node is produced, one can conclude that the sequent is provable classically.

(b) In Porgi's countermodels, child nodes always force strictly more subformulas of the formulas of the sequent than do their parents.

(c) In one of Porgi's countermodels, all nodes other than the root node force the formula φ.

Porgi can also handle minimal logic, is capable of generating typed lambda terms instead of natural deductions, and can display the subformulas of a sequent that are forced at each node of a Kripke countermodel.

Porgi is implemented in Standard ML (SML/NJ Version 0.93), but produces a UNIX command, `porgi`, which can be invoked from the shell. Porgi can be obtained—in both source and binary (SPARC/Solaris) forms—via WWW URL `http://www.cis.ksu.edu/~allen/porgi.html`.

2 Approach

Porgi uses a depth-first procedure based on the contraction-free, multi-succedent calculus LJT* of [Dyc92] to answer questions about the multi-succedent deducibility relation, $\vdash$. The search procedure used exploits rule invertibility (semi-invertibility in one case) in order to reduce the amount of backtracking that is done. Duplicate formulas are removed from a subgoal sequent's antecedent

[*]A slightly revised version of the paper that appears in the proceedings of the *CADE-13 Workshop on Proof Search in Type-Theoretic Languages*, Rutgers University, pp. 109–116, 1996. The single non-trivial change is footnoted in the text.

[†]This work was partially supported by NSF grant CCR-9302962.

and succedent before the subgoal is processed. Initially, this proof procedure is used to determine whether the formula φ is deducible from the set of assumptions Γ.

When the answer to this question is "yes", Porgi must find a minimally sized, normal natural deduction of φ from Γ. It synthesizes such a deduction in a bottom-up manner (starting from the leaves, working toward the root), restricting its attention to normal deductions whose formulas are subformulas of the formulas in $\Gamma \cup \{\varphi\}$ (this is justified by the subformula property of intuitionistic logic (see, e.g., Theorem 10.3.6 of [TvD88])). For the purposes of this inductive process, a normal deduction D_1 is considered to be *as good as* a normal deduction D_2 iff

(i) D_1's conclusion is the same as D_2's conclusion; and

(ii) every assumption of D_1 is also an assumption of D_2; and

(iii) D_1's size is less-than-or-equal-to D_2's size; and

(iv) if the conclusion of D_1 ends a segment (see Definition 10.1.8 of [TvD88]) that begins with an introduction-rule or false-elimination rule, then so does the conclusion of D_2 (i.e., if D_1 can't be the major-premiss of an elimination rule, then neither can D_2).

When a new (normal) deduction is synthesized, it is only kept if there is no existing deduction that is at least as good as it. And if the new deduction is better than some existing deductions, then those deductions are discarded. When the inductive process terminates, all deductions of φ whose assumptions are contained in Γ are collected together (some deductions of φ may have sets of assumptions that are proper subsets of Γ). A minimally sized element of this set of normal deductions is then selected as the answer.

When φ is not deducible from Γ, Porgi must find a finite, tree-based Kripke model whose root node forces the formulas in Γ but does not force φ. To do this, one could make use of one of the standard tableau-based algorithms for constructing countermodels to sequents [Fit69, Und94]. These algorithms involve loop-checking, which could be avoided by using an algorithm based on Pinto and Dyckhoff's Calculus for Refutation of Intuitionistic Propositions (CRIP) [PD95]. Unfortunately, all of these algorithms sometimes generate needlessly large models, yielding non-classical models when classical models would do, and producing models with children that force no more subformulas of $\Gamma \cup \{\varphi\}$ than do their parents.

Instead of trying to develop a tableau-based approach to synthesizing smaller models, I designed a countermodel generation algorithm that manipulates prime (saturated) theories whose formulas are subformulas of $\Gamma \cup \{\varphi\}$, where a set Δ of subformulas of $\Gamma \cup \{\varphi\}$ is a *prime theory*[1] iff (i) Δ doesn't contain f (falsity), except in the case of minimal logic, and (ii) if Ψ is a nonempty set of subformulas of $\Gamma \cup \{\varphi\}$ such that $\Delta \vdash \Psi$, then there is a $\psi \in \Psi$ such that $\psi \in \Delta$ (cf., Definition 2.6.2 of [TvD88]). The desired countermodel will be formed from a poset of prime theories $\mathcal{T}$ that is ordered by inclusion and has the following properties:

(a) $\mathcal{T}$ has a least element that contains all of the formulas in Γ but does not contain φ.

(b) If Δ is in $\mathcal{T}$ and $\psi_1 \to \psi_2$ is a subformula of $\Gamma \cup \{\varphi\}$ that is not in Δ, then there is a $\Delta' \in \mathcal{T}$ such that $\Delta \cup \{\psi_1\} \subseteq \Delta'$ but $\psi_2 \notin \Delta'$.

Porgi forms $\mathcal{T}$ in a demand-driven manner. If, as a subgoal, it needs to synthesize an element of $\mathcal{T}$ that includes all of the formulas in Π_1 but none of the formulas in Π_2, it starts with the formulas in Π_1, and then adds additional formulas as described below, always subject to the constraint that Π_2 not be deducible from the augmentations of Π_1. First, it *handles* as many implications as possible

[1]Revised definition. Note that prime theories are deductively closed, and if $\psi_1 \vee \psi_2$ is in a prime theory Δ, then either ψ_1 or ψ_2 is in Δ.

in the theory that it's building: either the implication must be included in the theory, or its left-hand-side must be included in the theory. It then produces a prime theory Δ by adding as many elements of $\Gamma \cup \{\varphi\}$ as possible to the theory being built. If Δ has no unhandled implications, then the subgoal has been achieved. Otherwise, for each unhandled implication $\psi_1 \rightarrow \psi_2$, there must be a prime theory $\Delta' \in \mathcal{T}$ such that $\Delta \cup \{\psi_1\} \subseteq \Delta'$ but $\psi_2 \notin \Delta'$. In the worst case, a distinct subgoal will have to be generated for each unhandled implication. However, Porgi minimizes the number of subgoals generated. E.g., if $\psi_1 \rightarrow \psi_2$ and $\psi_1' \rightarrow \psi_2'$ are both unhandled and $\Delta \cup \{\psi_1, \psi_1'\} \not\vdash \{\psi_2, \psi_2'\}$, then Porgi might generate a single subgoal to form a prime theory containing $\Delta \cup \{\psi_1, \psi_1'\}$ but not containing $\{\psi_2, \psi_2'\}$. Many of the prime theories thus generated will fail to be *immediate* successors of Δ in the final version of $\mathcal{T}$. Note that a given element of $\mathcal{T}$ may be formed multiple times as $\mathcal{T}$ is built.

Finally, $\mathcal{T}$ must be turned into a tree-based Kripke model. $\mathcal{T}$ is made into a tree labeled by the elements of $\mathcal{T}$ in the obvious way: if a given node is labeled by the theory Δ, and $\Delta_1, \ldots, \Delta_n$ are the immediate successors of Δ in $\mathcal{T}$, then this node will have n children with labels $\Delta_1, \ldots, \Delta_n$. The resulting tree may have multiple nodes labeled with identical theories; but there will never be two such nodes where one is an ancestor of the other. Only the root node will lack φ. Any redundant non-root nodes of the tree are then removed, one by one, where such a node is *redundant* iff it can be removed while preserving the following tree-oriented restatement of property (b):

(b') If Δ is a node's label and $\psi_1 \rightarrow \psi_2$ is a subformula of $\Gamma \cup \{\varphi\}$ that is not in Δ, then there is a descendant of the node whose label contains ψ_1 but does not contain ψ_2.

(When a node is removed from the tree, its children become the children of its parent.) The resulting tree is made into a Kripke model by stipulating that a given node of the tree forces exactly those propositional variables that are contained in the prime theory that is the node's label. As a consequence, a subformula of $\Gamma \cup \{\varphi\}$ is forced by a node iff the subformula is an element of the theory that is the node's label.

3 Examples

Figure 1 shows Porgi's response to the `--help` command line option.

Figure 2 shows how Porgi can be used to find a countermodel $\mathcal{A}$ to the formula $\varphi = (\neg\neg P \rightarrow P) \vee \neg P \vee \neg\neg P$. (The letters A–Z are propositional variables; the nodes of one of Porgi's countermodels are numbered in the order that they would be reached in a preorder traversal of the countermodel tree; and all timings are on a SPARCstation 10.) The traditional presentation of $\mathcal{A}$ is given in Figure 3, and the output of Figure 4 lists the subformulas of φ that are forced at each of $\mathcal{A}$'s nodes. $\mathcal{A}$ has one less node than the countermodel for φ given on p. 79 of [TvD88].

On the other hand, φ becomes provable when $P \vee \neg P$ is allowed as an assumption, as can be seen by examining Figure 5, whose output consists of a natural deduction of φ from $P \vee \neg P$. This natural deduction is displayed in the traditional manner in Figure 6, and the corresponding lambda term is the output of Figure 7.

Figure 8 shows how Porgi can be used to check that the sequent $\neg\neg A \rightarrow \neg\neg B \Rightarrow \neg\neg(A \rightarrow B)$ is provable (i.e., without going to the additional work of generating a minimally sized deduction of $\neg\neg(A \rightarrow B)$ from $\neg\neg A \rightarrow \neg\neg B$). On the other hand, this sequent is not provable in minimal logic, and the result of Figure 9 consists of a minimal logic countermodel of the sequent.

The interesting thing about the countermodel to $\psi = (A \rightarrow B) \vee (B \rightarrow C) \vee (C \rightarrow D) \vee (D \rightarrow E) \vee$

Figure 1: `porgi`'s response to the `--help` option

```
% porgi --help
Porgi Version 1.0

Usage: porgi [OPTION]... sequent

  -m, --minimal  use minimal logic
  -c, --check    just check deducibility
  -l, --lambda   output lambda term instead of deduction
  -v, --verbose  output verbose results
  -t, --trace    output tracing information
  -h, --help     output this information and exit

Note: "sequent" must be a single argument, and should have the form

    form, ..., form => form

where each "form" is a propositional formula built up using parentheses,
whitespace and the following symbols:

    f   (falsity)
    A-Z (propositional variables)
    ~   (negation)       highest precedence
    &   (conjunction)       .                 right associative
    |   (disjunction)       .                 right associative
    ->  (implication)       .                 right associative
    <-> (biimplication)  lowest precedence    right associative
```

Figure 2: a countermodel of $\Rightarrow (\neg\neg P \to P) \vee \neg P \vee \neg\neg P$

```
% porgi "=> (~~P -> P) | ~P | ~~P"
unprovable: => (~~P -> P) | ~P | ~~P

        2: { P }
    1: {  }
    3: {  }
0: {  }

[0.010 sec user cpu time (0.010 sec non-gc, 0.000 sec gc)]
```

Figure 3: the traditional presentation of the Kripke model of Figure 2

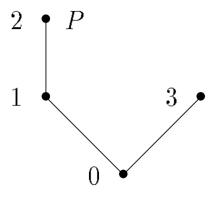

Figure 4: the subformulas forced by the nodes of Figure 2's model

```
% porgi --verbose "=> (~~P -> P) | ~P | ~~P"
unprovable: => (~~P -> P) | ~P | ~~P

        2: { P,
             ~~P,
             ~P | ~~P,
             (~~P -> P) | ~P | ~~P,
             ~~P -> P }
    1: { ~~P,
         ~P | ~~P,
         (~~P -> P) | ~P | ~~P }
    3: { ~P,
         ~P | ~~P,
         (~~P -> P) | ~P | ~~P,
         ~~P -> P }
0: {  }

[0.010 sec user cpu time (0.010 sec non-gc, 0.000 sec gc)]
```

Figure 5: a natural deduction of $P \vee \neg P \Rightarrow (\neg\neg P \to P) \vee \neg P \vee \neg\neg P$

```
% porgi "P | ~P => (~~P -> P) | ~P | ~~P"
provable: P | ~P => (~~P -> P) | ~P | ~~P

    Ass: P | ~P
            (1) P
        ->I (0): ~~P -> P
    |I1: (~~P -> P) | ~P | ~~P
            (1) ~P
        |I1: ~P | ~~P
    |I2: (~~P -> P) | ~P | ~~P
|E (1): (~~P -> P) | ~P | ~~P

[0.410 sec user cpu time (0.410 sec non-gc, 0.000 sec gc)]
```

Figure 6: the traditional presentation of the natural deduction of Figure 5

$$\dfrac{P \vee \neg P \qquad \dfrac{\dfrac{P\ (1)}{\neg\neg P \to P}\ {\to}\mathrm{I}\ (0)}{(\neg\neg P \to P) \vee \neg P \vee \neg\neg P}\ \vee\mathrm{I1} \qquad \dfrac{\dfrac{\neg P\ (1)}{\neg P \vee \neg\neg P}\ \vee\mathrm{I1}}{(\neg\neg P \to P) \vee \neg P \vee \neg\neg P}\ \vee\mathrm{I2}}{(\neg\neg P \to P) \vee \neg P \vee \neg\neg P}\ \vee\mathrm{E}\ (1)$$

Figure 7: the lambda term corresponding to Figure 5's deduction

```
% porgi --lambda "P | ~P => (~~P -> P) | ~P | ~~P"
provable: x0: P | ~P => (~~P -> P) | ~P | ~~P

case y1: P, ~P
    x0
    in1 ~P | ~~P
        lambda y0: ~~P
            y1
    in2 ~~P -> P
        in1 ~~P
            y1

[0.350 sec user cpu time (0.350 sec non-gc, 0.000 sec gc)]
```

Figure 8: checking that $\neg\neg A \to \neg\neg B \Rightarrow \neg\neg(A \to B)$ is provable

```
% porgi --check "~~A -> ~~B => ~~(A -> B)"
provable: ~~A -> ~~B => ~~(A -> B)

[0.010 sec user cpu time (0.010 sec non-gc, 0.000 sec gc)]
```

Figure 9: a minimal logic countermodel to the sequent of Figure 8

```
% porgi --minimal "~~A -> ~~B => ~~(A -> B)"
unprovable: ~~A -> ~~B => ~~(A -> B)

    1: { f, A }
0: {  }

[0.160 sec user cpu time (0.160 sec non-gc, 0.000 sec gc)]
```

$(E \to F) \vee (F \to A)$ of Figure 10 is that nodes 1 and 2 both have three roles: node 2 forces A but not B, forces C but not D, and forces E but not F; and node 1 forces B but not C, forces D but not E, and forces F but not A. The more obvious countermodel of ψ has six nodes in addition to its root.

The output of Figure 11 consists of a countermodel to an example suggested by Roy Dyckhoff. This model has four fewer nodes than the countermodel generated by his and Pinto's CRIP-based system [Dyc95].

It is easy to find sequents that take Porgi unacceptably long to prove or refute. For example, it takes Porgi 104 seconds to generate a minimal deduction of the sequent $\neg\neg A \vee \neg\neg B \vee \neg\neg C \Rightarrow \neg\neg(A \vee B \vee C)$. A countermodel generation problem that is hard for Porgi to solve can be found in Figure 12. The reader will have no trouble coming up with sequents that Porgi will have even more trouble with.

Finally, Figure 13 shows that Porgi can fail to produce countermodels that are minimally sized.

Figure 10: a countermodel to $(A \to B) \vee (B \to C) \vee (C \to D) \vee (D \to E) \vee (E \to F) \vee (F \to A)$

```
% porgi "=> (A -> B) | (B -> C) | (C -> D) | (D -> E) | (E -> F) | (F -> A)"
unprovable: => (A -> B) | (B -> C) | (C -> D) | (D -> E) | (E -> F) | (F -> A)

    1: { B, D, F }
    2: { A, C, E }
0: {  }

[0.050 sec user cpu time (0.050 sec non-gc, 0.000 sec gc)]
```

Figure 11: a countermodel to $(((\neg\neg P \to P) \to P \vee \neg P) \to \neg P \vee \neg\neg P) \to \neg\neg P \vee (\neg\neg P \to P)$

```
% porgi "=> (((~~P -> P) -> P | ~P) -> ~P | ~~P) -> ~~P | (~~P -> P)"
unprovable: => (((~~P -> P) -> P | ~P) -> ~P | ~~P) -> ~~P | (~~P -> P)

        2: {  }
        3: { P }
    1: {  }
        5: { P }
    4: {  }
0: {  }

[0.110 sec user cpu time (0.110 sec non-gc, 0.000 sec gc)]
```

Figure 12: "hard" countermodel generation example

```
% porgi "=> ~~A | (A -> ~~B | (B -> ~~C | (C -> (~~D -> D) | ~D | ~~D)))"
unprovable: => ~~A | (A -> ~~B | (B -> ~~C | (C -> (~~D -> D) | ~D | ~~D)))

                    5: { A, B, C, D }
                4: { A, B, C }
                6: { A, B, C }
            3: { A, B, C }
            7: { A, B }
        2: { A, B }
        8: { A }
    1: { A }
    9: {  }
0: {  }

[68.430 sec user cpu time (67.310 sec non-gc, 1.120 sec gc)]
```

Figure 13: failure to generate a minimally sized countermodel

```
% porgi "=> (~~P -> P) | ~P | ~~P | (~~Q -> Q) | ~Q | ~~Q"
unprovable: => (~~P -> P) | ~P | ~~P | (~~Q -> Q) | ~Q | ~~Q

        2: { P, Q }
    1: { P }
    3: {  }
        5: { P, Q }
    4: { Q }
0: {  }

[0.240 sec user cpu time (0.240 sec non-gc, 0.000 sec gc)]
```

A smaller countermodel of this figure's sequent can be constructed from the model of Figure 3 by stipulating that Q as well as P is forced by node 2.

4 Directions for further research

The inefficiency of Porgi's algorithm for generating minimally sized, normal natural deductions limits its utility. And, Porgi's algorithm for generating Kripke countermodels has two drawbacks: (i) it doesn't always generate minimally sized countermodels, and (ii) it is much less efficient than the existing tableau-based algorithms for countermodel generation.

Ideally, one would like to have a system that could efficiently generate minimally sized deductions and countermodels. But, given the choice between *efficiency* and *minimality*, I would choose minimality. My current priority is the development of a usable algorithm for generating minimally sized Kripke countermodels.

Acknowledgments

It is a pleasure to acknowledge helpful conversations and/or e-mail exchanges with Roy Dyckhoff, Brian Howard, Andy Pitts, David Schmidt, Colin Stirling and Lincoln Wallen.

References

[Dyc92] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992.

[Dyc95] R. Dyckhoff. Private communication via e-mail, 1995.

[Fit69] M. C. Fitting. *Intuitionistic Logic, Model Theory and Forcing*. North-Holland, 1969.

[PD95] L. Pinto and R. Dyckhoff. Loop-free construction of counter-models for intuitionistic propositional logic. In Behara, Fritsch, and Lintz, editors, *Symposia Gaussiana*, pages 225–232. Walter de Gruyter, 1995.

[TvD88] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1988.

[Und94] J. Underwood. *Aspects of the Computational Content of Proofs*. PhD thesis, Cornell University, 1994.