Abductive Intuitionistic Propositional Logic as
Theorem Synthesis

Paul Tarau
University of North Texas
paul.tarau@unt.edu

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

After introducing a compact Prolog-based theorem prover for Intuitionistic Propositional Logic, we design
with its help a protasis generator that synthesizes, given a formula, minimal assumptions under which the
formula becomes a theorem. Besides implementing on top of it the usual abductive reasoning mechanisms,
we parameterize our protasis generator with the binary connectives available in the logic. As special cases,
we synthesize conjunctions of literals that mimic rows of truth tables in classical or intermediate logics and
we abduce conditional hypotheses that turn theorems in classical or intermediate logics into theorems in
intuitionistic logic.

One step further, we generalize abductive reasoning to synthesize sequent premises using a minimal
canonical set of formulas equiprovable in the underlying calculus.

Organized as a self-contained literate Prolog program, the paper supports interactive exploration of its
content and ensures full replicability of our results.

Keywords: abductive intuitionistic logic, theorem synthesis, logic programming and automated reason-
ing, theorem provers for intuitionistic propositional logic, implementing sequent calculi in Prolog.

1 Introduction

Given a formula $F$ in Classical Propositional Logic (CL), each row in a formula’s truth table
describes a conjunction of literals $C$. Reading the truth table as a disjunctive normal form, it
immediately follows that $C \rightarrow F$ is a tautology. As an example, let us consider the CL formula
$F = (A \lor B) \& (B \lor C) \& (C \lor A)$. Then its truth table (with 1 for True and 0 for False) is
the one on the left. Let us select any row, say [1,0,1] and interpret it as $G = A \& \neg B \& C$. Then
the truth table of the resulting tautology $G \rightarrow F$ is shown on the right.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>G-&gt;F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus, it is easy to find, for a formula $F$ in CL, a set of assumptions which make it a theorem
in CL.
This model-theoretic approach extends also to intermediate logics\(^1\) and in particular, it applies to the 5-valued truth-tables of the equilibrium logic (Pearce et al. 2000), underlying Answer Set Programming (ASP).

With no finite truth-tables, no inter-definability of logical connectives, no rule of excluded middle and only a concept of tautology and contradiction defined for Intuitionistic Propositional Logic (IL), we need to be a bit more creative when trying to find salient assumptions that would make the formula a theorem in IL. First, given a formula in IL, we will need a search process for finding assumptions that would make it a theorem. Next, we would like our assumptions to be minimal with respect to the partial order relation governing the logic (or its equivalent Heyting algebra), intuitionistic implication.

This brings us to Abductive Logic Programming (Eshghi and Kowalski 1989; Denecker and Kakas 2002), where facts designated as abducibles are filtered with integrity constraints to provide relevant assumptions needed for the success of a goal \(G\) w.r.t. a given program \(P\). In the context of IL, our abductive reasoning will rely on finding minimal assumptions under which a formula becomes a theorem.

And finally, in the absence of a convenient automated semantic method like truth tables or SAT solvers in CL, we will need a theorem prover, ideally derived directly from the rules of a terminating sequent calculus, that interoperates smoothly with the search process synthesizing our assumptions.

These requirements make Prolog a natural meta-language for an actionable description of these concepts. We will materialize our approach as a literate Prolog program, from which, as a convenience to the reader, we will extract our code and make it available online as a Prolog file\(^2\).

The rest of the paper is organized as follows. Section 2 overviews our Prolog-based theorem prover and the sequent calculus it is derived from. Section 3 introduces our protasis synthesizer and its uses for abductive reasoning. Section 4 generalizes our approach to the synthesis of minimal canonical assumptions. Section 5 discusses significance of our results, its possible extensions as well as some of its limitations. Section 6 overviews related work and section 7 concludes the paper.

We assume the reader is fluent in Prolog, propositional intuitionistic and classical logic and familiar with abductive reasoning and key concepts behind sequent calculi and automated theorem proving.

## 2 The Prover

We will derive our Prolog prover from a set of compact and elegant sequent calculus rules formally describing provability in IL.

### 2.1 Roy Dyckhoff’s G4ip calculus

Motivated by problems related to loop avoidance in implementing Gentzen’s LJ calculus, Roy Dyckhoff designed and proved sound and complete a sequent calculus-based axiomatization of IL (Dyckhoff 1992). He has proved that the calculus is terminating, by identifying a multiset ordering-based formula size definition that decreases after each step (Dyckhoff 1992).

\(^1\) logics weaker than classical but stronger than intuitionistic

\(^2\) at https://github.com/ptarau/TypesAndProofs/blob/master/isynt.pro
The sequents of the G4ip calculus follow:

\[
\begin{align*}
\Gamma, p & \Rightarrow p \quad \text{(p an atom)} \\
\Gamma & \Rightarrow \phi \quad \Gamma & \Rightarrow \psi \\
\frac{}{\Gamma \Rightarrow \phi \land \psi} \quad R \land & \\
\frac{}{\Gamma, \phi \land \psi \Rightarrow \Delta} \quad L \land & \\
\Gamma & \Rightarrow \phi_i \\
\frac{}{\Gamma \Rightarrow \phi_0 \lor \phi_1} \quad R \lor & \\
\frac{}{\Gamma, \phi \lor \psi \Rightarrow \Delta} \quad L \lor & \\
\Gamma, \phi \Rightarrow \psi \\
\frac{}{\Gamma \Rightarrow \phi \Rightarrow \psi} \quad R \Rightarrow & \\
\frac{}{\Gamma, \phi \Rightarrow \psi \Rightarrow \gamma \Rightarrow \Delta} \quad L \Rightarrow & \\
\Gamma, \phi \rightarrow (\psi \rightarrow \gamma) & \Rightarrow \Delta \\
\Gamma, \phi \land \psi \rightarrow \gamma & \Rightarrow \Delta \quad L \land \rightarrow & \\
\Gamma, \phi \lor \psi \rightarrow \gamma & \Rightarrow \Delta \quad L \lor \rightarrow & \\
\Gamma, \psi \rightarrow \gamma \Rightarrow \phi \rightarrow \psi \\
\Gamma, \phi \rightarrow \psi \rightarrow \gamma, \gamma \Rightarrow \Delta & \Rightarrow \Delta \quad L \rightarrow \rightarrow & \\
\end{align*}
\]

Key to the termination proof in (Dyckhoff 1992) is the rule \( L \rightarrow \rightarrow \) that breaks down nested implications into “smaller” ones, each containing fewer connectives. The rules work with the context \( \Gamma \) being a multiset, but it has been shown later (Dyckhoff 2016) that \( \Gamma \) can be a set, with duplication in contexts eliminated. Note that the same calculus has been discovered independently in the 50’s by Vorob’ev and in the 80’s-90’s by Hudelmaier (Hudelmaier 1988).

### 2.2 Implementing the Theorem Prover

In the tradition of "lean theorem provers”, we can build one directly from the G4ip calculus, in a goal oriented style, by reading the rules from conclusions to premises.

Thus, we start with a simple, almost literal translation of sequent rules to Prolog with values in the environment \( \Gamma \) denoted by the variable \( V_s \). Besides implication (denoted \( \rightarrow \)), conjunction (denoted \( \& \)) and disjunction (denoted \( \lor \)), we implement rules for inverse implication (denoted \( \leftarrow \)), negation (denoted \( \neg \)) and equivalence (denoted \( \leftrightarrow \)). We also add rules for a top element (denoted \( \text{true} \)) and a bottom element (denoted \( \text{false} \)). Correctness of our additions follows from the definitions of:

- intuitionistic inverse implication: \( \phi \leftarrow \psi \equiv \psi \rightarrow \phi \)
- intuitionistic equivalence: \( \phi \leftrightarrow \psi \equiv \phi \rightarrow \psi \land \psi \rightarrow \phi \)
- intuitionistic negation: \( \neg \phi \equiv \phi \rightarrow \text{false} \)
- top element: \( \text{true} \equiv \text{false} \rightarrow \text{false} \).

We define operators for all our connectives, except \( \rightarrow \), that we will use with its standard right associativity.

```prolog
:- op(525, fy, ~ ).
:- op(550, xfy, & ).  % right associative
:- op(575, xfx, \lor ). % right associative
:- op(600, xfx, <-> ). % non associative
:- op(800, yfx, <\-
```

- . % left associative
A formula $T$ is a theorem if it is provable from an empty set of assumptions:

```
iprover(T) :- iprover(T, []).  
```

We follow here Dyckhoff’s calculus but delegate details to helper predicates `iprover_reduce/4` and `iprover_impl/4`.

```
iprover(true, _):-!.
iprover(A, Vs):=-memberchk(A, Vs),!.
iprover(\^A, Vs):=-iprover(false, [A|Vs]).
iprover(A\rightarrow B, Vs):=-!,iprover(B, [A|Vs]),iprover(A, [B|Vs]).
iprover((A\rightarrow B), Vs):=-!,iprover(B, [A|Vs]).
iprover((B<-A), Vs):=-!,iprover(B, [A|Vs]).
iprover(A \& B, Vs):=-!,iprover(A, Vs),iprover(B, Vs).
iprover(G, Vs):-%
  atomic or disj or false
  select(Var, Vs, Vs1),
  iprover_reduce(Var, G, Vs1, Vs2),
  !,
  iprover(G, Vs2).
iprover(A \lor B, Vs):=(iprover(A, Vs) ; iprover(B, Vs)),!.
```

`iprover_reduce/4` is the first step in breaking down formulas in the premise into their components.

```
iprover_reduce(true, _, Vs1, Vs2):=-!,iprover_impl(false, false, Vs1, Vs2).
iprover_reduce(\^A, _, Vs1, Vs2):=-!,iprover_impl(A, false, Vs1, Vs2).
iprover_reduce((A\rightarrow B), _, Vs1, Vs2):=-!,iprover_impl(A, B, Vs1, Vs2).
iprover_reduce((B<-A), _, Vs1, Vs2):=-!,iprover_impl(A, B, Vs1, Vs2).
iprover_reduce((A \& B), _, Vs, [A,B|Vs]):=-!.
iprover_reduce((A\rightarrow B), _, Vs, [(A\rightarrow B), (B<-A)|Vs]):=-!.
iprover_reduce((A \lor B), G, Vs, [B|Vs]):=iprover(G, [A|Vs]).
```

`iprover_impl/4` details the case analysis of the handling of implication (and its instances) prescribed by rule $L \rightarrow \rightarrow$.

```
iprover_impl(true, B, Vs, [B|Vs]):=-!.
iprover_impl(\^C, B, Vs, [B|Vs]):=-!,iprover_impl(C\rightarrow false, Vs).
iprover_impl((C\rightarrow D), B, Vs, [B|Vs]):=-!,iprover_impl((C\rightarrow D), [(D\rightarrow B)|Vs]).
iprover_impl(D\rightarrow C, B, Vs, [B|Vs]):=-!,iprover_impl((C\rightarrow D), [(D\rightarrow B)|Vs]).
iprover_impl((C \& D), B, Vs, [(C\rightarrow D)|Vs]):=-!.
iprover_impl((C \lor D), B, Vs, [(C\rightarrow B), (D\rightarrow B)|Vs]):=-!.
iprover_impl((C\rightarrow D), B, Vs, [(C\rightarrow D)\rightarrow((D\rightarrow C)\rightarrow B)|Vs]):=-!.
iprover_impl(A, B, Vs, [B|Vs]):=memberchk(A, Vs).
```

**Example 1**

After defining:

```
iprover_test:-
  Taut = ((p \& q) \leftrightarrow (((p \lor q)\leftrightarrow q)\leftrightarrow p)), iprover(Taut),
  Contr=(a \& \^a), \dag (iprover(Contr)).
```

we observe success on proving a tautology and failing to prove a contradiction, but we refer to
Glivenko’s theorem states that a propositional formula \( F \) is a classical tautology if and only if \( \neg \neg F \) is an intuitionistic tautology. This gives us a classical prover for free, that we will use as an alternative to \texttt{iprover} when defining several concepts parameterized by a prover.

\begin{verbatim}
% classical prover - via Glivenko's theorem
cprover(T):-iprover( ~ ~T).
\end{verbatim}

\textbf{Example 2}

The two provers, as it is well known, will disagree on \( p \lor \neg p \)

\begin{verbatim}
?- iprover(p \lor \neg p).
false.
?- cprover(p \lor \neg p).
true.
\end{verbatim}

but agree on \( p \land \neg p \):

\begin{verbatim}
?- iprover(p \land \neg p).
false.
?- cprover(p \land \neg p).
false.
\end{verbatim}

\section{Abductive Reasoning Mechanisms}

We are now ready to introduce our search for assumptions that make a given formula an intuitionistic tautology.

\subsection{Generating the Abducibles}

Defining some of the atoms occurring in a formula \( F \) as the only ones to be used in the search process brings us to declare them as abducibles (Eshghi and Kowalski 1989), but we will also enable the option to make abducible all the atoms occurring in \( F \). Thus, when the variable Abducibles is free, all atomic symbols will be considered abducibles.

\begin{verbatim}
abducibles_of(Formula,Abducibles):-var(Abducibles),!,atoms_of(Formula,Abducibles).
abducibles_of(_,_).
\end{verbatim}

The predicate \texttt{atoms_of/2} finds the set of all the atoms occurring in a formula. It backtracks over all arguments, recursively, it eliminates duplicates with SWI-Prolog’s built-in \texttt{distinct/2} and it collects them to a list with \texttt{findall/3}.

\begin{verbatim}
atom_of(A,R):-atomic(A),!,R=A.
atom_of(T,A):-arg(_,T,X),atom_of(X,A).
atoms_of(T,As):-findall(A,distinct(A,atom_of(T,A)),As).
\end{verbatim}

\footnote{http://www.iltp.de/}
If $F$ is a formula, we can think of a premise $C$ such that $C \rightarrow F$ is a theorem as a counterfactual assumption making $F$ conditionally true. We call such a formula $C$ a protasis. Thus, given a formula and a set of assumptions (our abducibles), we will need to search among them for assumptions that would make the formula a theorem.

The predicate any_protasis/6 implements this idea, subject to a set of parameters:

- **Prover** allows a choice of the underlying logic (e.g., intuitionistic or classical)
- **AggregatorOp** fixes one of the connectives of the logic, from which the protasis is built
- **The yes/no flag WithNeg decides if negations of the abducibles can be part of the protasis**
- **Abducibles** is a set of atoms or a free variable, meaning that all atoms will be included
- **Assumption** will be any protasis, that given the previously specified parameters, ensures that $\text{Assumption}\rightarrow\text{Formula}$ is a theorem in the logic specified by Prover.

```prolog
any_protasis(Prover,AggregatorOp,WithNeg,Abducibles,Formula,Assumption):-  
  abducibles_of(Formula,Abducibles),  
  mark_hypos(WithNeg,Abducibles,Literals),  
  subset_of(Literals,Hypos),  
  join_with(AggregatorOp,Hypos,Assumption),  
  \+ (call(Prover,Assumption->false)), % we do not assume contradictions !  
  call(Prover,Assumption->Formula). % we ensure this is a theorem
```

The predicate mark_hypos/3 will mark with their negations the abducibles if we want to allow them to occur in the protasis positively or prefixed by their negations.

```prolog
mark_hypos(_,[],[]).
mark_hypos(yes,[P|Ps],[P,~P|Ns]):-mark_hypos(yes,Ps,Ns).
mark_hypos(no,[P|Ps],[P|Ns]):-mark_hypos(no,Ps,Ns).
```

Our subset generator subset_of, to be used to iterate over all subsets of the abducibles, first enumerates templates of increasing length as we want smaller subsets to be tried first.

```prolog
subset_of(Xs,Ts):-template_from(Xs,Ts),tsubset(Xs,Ts).
```

```prolog
template_from(_,[]).
template_from([_|Xs],[_|Zs]):-template_from(Xs,Zs).
```

Then, for each template of length $K$, it fills it with a subset of length $K$ of the $N$ abducibles, one at a time, on backtracking.

```prolog
tsubset([],[]).
tsubset([X|Xs],[X|R]):-tsubset(Xs,R).
tsubset([_|Xs],R):-tsubset(Xs,R).
```

The predicate join_with_op builds an expression from a sequence of abducible literals (atoms, possibly negated) with a given operator.

```prolog
join_with_op(_,[],true).
join_with_op(_,[],X).
join_with_op(Op,[X|Y|Xs],R):-join_with_op(Op,[Y|Xs],R0),R=..[Op,X,R0].
```

---

4 or, equivalently, when $C$ is the premise and $F$ is the conclusion of a provable sequent
How we join the abducible literals with help of a given operator, is different for associative and
commutative operators like \& and v (the default 3-rd clause of \texttt{join} with) and \( \rightarrow, \leftarrow, \leftrightarrow \), that
we treat as special cases.

Thus, once the head was picked with \texttt{select/3}, the order of the remaining literals is immaterial.

\begin{verbatim}
join_with(Op,Xs,R):-
    memberchk(Op,[(\rightarrow),((\leftarrow))],!),
    select(Head,Xs,YS), append(YS,[Head],ZS),
    join_with_op((\rightarrow),ZS,R).
\end{verbatim}

For non-associative \( \leftrightarrow \) we will use all permutations of the abducibles.

\begin{verbatim}
join_with(Op,Xs,R):-Op=(\leftrightarrow),!,permutation(Xs,YS),join_with_op(Op,YS,R).
\end{verbatim}

Finally, as v and \& are permutation invariant, we just call \texttt{join_with_op}.

\begin{verbatim}
join_with(Op,Xs,R):- join_with_op(Op,Xs,R).
\end{verbatim}

Implication and reverse implications are handled in \texttt{join} with, knowing that their components,
extcept the head, are permutation invariant, with the following equivalences in mind:

\[ \phi_0 \leftarrow \phi_1 \cdots \leftarrow \phi_n \equiv \phi_0 \leftarrow \phi_1 \& \cdots \& \phi_n \]

and

\[ \phi_n \rightarrow \phi_{n-1} \cdots \rightarrow \phi_1 \rightarrow \phi_0 \equiv \phi_n \& \phi_{n-1} \& \cdots \& \phi_1 \rightarrow \phi_0 \]

Note that these hold both intuitionistically and classically, as it can be quickly verified with
\texttt{iprover} and \texttt{cprover}.

3.3 The Weakest Protasis

The next step is defining a \textit{weakest protasis}, keeping in mind that a \textit{partial order}\footnote{in contrast to classical logic, where \((p\rightarrow q) \lor (q\rightarrow p)\) is a theorem} among them
is defined by the intuitionistic implication \(\rightarrow\). First, we will collect with \texttt{findall/3} all the
candidate assumptions to ensure that the prover is called on each only once.

\begin{verbatim}
weakest_protasis(Prover,AggregatorOp,WithNeg,Abducibles,Formula,Assumption):-
    findall(Assumption,
        any_protasis(Prover,AggregatorOp,WithNeg,Abducibles,Formula,Assumption),
        Assumptions),
    weakest_with(Prover,Assumptions,Assumption).
\end{verbatim}

Next, we ensure that a \textit{weakest protasis} is such that it does not imply any other protasis, thus that
it is a minimal element w.r.t. our partial order. We rely for that on the predicate \texttt{weakest_with/3}:

\begin{verbatim}
weakest_with(_,Gs,G):-memberchk(true,Gs),!,G=true.
weakest_with(Prover,Gs,G):-select(G,Gs,Others),
    \(+\ member(Other,Others),weaker_with(Prover,Other,G)).
\end{verbatim}

The partial order relation is exposed as the predicate \texttt{weaker_with/3} which ensures that a weak-
est protasis does not imply any other protasis, including ones that might imply it.
weaker_with(Prover,P,Q):- \+ call(Prover,(P->Q)), call(Prover,(Q->P)).

Note that the predicate weakest_protasis/6 depends on the same parameters as any_protasis, in particular on the Prover implementing a given provability relation.

**Example 3**

Peirce’s law is known to hold in CL and not hold in IL. In fact, it can turn IL into CL if added as an axiom. The predicate peirce/2 will try to synthesize an assumption that would make it hold.

```
peirce(Prover,WhatIf):-
    Formula=((p->q)->p)->p),
    WithNeg=yes, AggregatorOp=(v), Abducibles=[p],
```

When running it we get:

```
?- peirce(iprover,Protasis).
Protasis = p v ~p.
?- peirce(cprover,Protasis).
Protasis = true.
```

This reveals an interesting fact. It tells us that if p v ~p were assumed, Peirce’s law would hold in IL, as iprove would succeed. As it is well known, p v ~p would turn IL into CL and cprove tells us that indeed, Peirce’s law holds unconditionally in CL.

**Example 4**

We can also synthesize conditional assumptions when using implication or inverse implication as our aggregator connective. After defining:

```
impl_aggr(H):-
    T=(a<-((a<-b<-d) & (b<-c))),
    Prover=iprover, WithNeg=yes, AggregatorOp=(->), As=[c,d],
    weakest_protasis(Prover,AggregatorOp,WithNeg,As,T,H).
```

we obtain:

```
?- impl_aggr(H).
H = ( d->c).
```

showing that the implication d->c (equivalent of the Horn Clause c:-d) should hold for the formula to be a theorem. This illustrates a mechanism to synthesize Horn Clauses playing the role of conditional assumptions.

Another interesting case is that of a contradiction. After defining contra_test/1 as:

```
contra_test(H):-
    T=([p & ~p),
    Prover=iprover, WithNeg=yes, AggregatorOp=(&),
    weakest_protasis(Prover,AggregatorOp,WithNeg,Abducibles,T,H).
```

and running it with:

```
?- contra_test(Protasis).
false.
```
we can see that no assumption would make the contradiction a tautology, and this will also be
the case for Prover=cprover. Note that to ensure this, we have enforced in the definition of
any_protasis that such assumptions should themselves not be contradictions.

Example 5
In the case of the logic of here-and-there (Pearce 1997), derived from IL by adding the axiom
\( f \lor (f \rightarrow g) \lor \neg g \)
as shown in in (Lifschitz et al. 2001), we will get with cprover the protasis true. This indicates,
as expected, that it is already a theorem in CL and thus also a theorem in the logic of here-and-
there.

\[
?\text{- weakest_protasis(cprover,(v),yes,_,(f \lor (f \rightarrow g) \lor \neg g),P).}
P = \text{true.}
\]

On the other hand, the less obvious weakest protasis obtained for iprover indicates that the
excluded middle rule would be needed for both \( f \) and \( g \).

\[
?\text{- weakest_protasis(iprover,(v),yes,_,(f \lor (f \rightarrow g) \lor \neg g),P).}
P = f \lor \neg f \lor g \lor \neg g.
\]

3.4 Implementing Intuitionistic Abductive Reasoning
With the logic of synthesizing meaningful minimal assumptions that make a formula a theorem
clarified, we are now ready to revisit abductive reasoning along the lines of (Eshghi and Kowalski
1989).

The predicate explain_with/5 finds, given a Prover, the abductive inference problem pa-
rameterized by:

- a formula Prog seen here as representing a knowledge base
- a set of Abducibles occurring in Prog
- a goal formula G such that Prog \( \rightarrow \) G should always hold
- a formula IC playing the role of integrity constraints meant to filter out unwanted assump-
tions

\[
\text{explain_with(Prover,Abducibles,Prog,IC,G,Expl):-}
\text{any_protasis(Prover,(&),yes,Abducibles,(Prog\rightarrow G), Expl),}
\text{call(Prover, Expl & Prog\rightarrow G),}
\text{call(Prover,(Expl & Prog\rightarrow IC)),}
\text{\(+ (call(Prover,(Expl & Prog \rightarrow false))).\)}
\]

Note also that any_protasis replaces weakest_protasis in this definition, given that mini-
mality might want to be stated as part of the integrity constraints IC.

Example 6
To revisit a simple example of abductive explanation generation, we define:

\[
\text{why_wet(Prover):-}
\text{IC = \neg(rained & sunny),}
\text{P = sunny \& (rained \lor sprinkler \rightarrow \neg wet), As=[sprinkler,rained], G = wet,}
\text{writeln(prog=P), writeln(ic=IC),}
\text{explain_with(Prover,As,P,IC,G,Explanation),}
\text{writeln('Explanation:' \rightarrow Explanation).}
\]
Paul Tarau

Then, when running it, it will display, as expected:

```prolog
?- why_wet(irover).
prog=sunny&(rained v sprinkler->wet)
ic= ¬ (rained&sunny)
Explanation: --> sprinkler& ¬rained
```

4 Synthesis of Minimal Canonical Assumptions

We will now generalize our abductive reasoning mechanism by lifting the constraint on the premise of our sequent from literals connected by a single operation to a canonical form that has been shown to be able to represent arbitrary IL formulas.

4.1 The Mints Transformation

Grigori Mints has proven, in his seminal paper studying complexity classes for intuitionistic propositional logic (Mints 1992), that a formula $f$ is equiprovable to a formula of the form $X_f \rightarrow g$ where $X_f$ is a conjunction of formulas of one of the forms:

\[ p, \neg p, p \rightarrow q, (p \rightarrow q) \rightarrow r, p \rightarrow (q \rightarrow r), p \rightarrow (q v r), p \rightarrow \neg q, \neg q \rightarrow p. \]

With introduction of new variables (like with the Tseitin transform for SAT or ASP solvers), the transformation runs in linear space and time. Note that as a premise to a sequent can be seen as a conjunction implying its conclusion, the conjunction of the formulas described by Mints can be seen as serving the same purpose as the conjunctive normal form (CNF) in classical logic.

Thus, by generating this set of bounded size formulas as premise of a sequent, we can express equivalent formulas of unbounded size, otherwise subject to a much larger search space in the formula synthesis process.

We will now generate, using a given set of abducibles, premises built of these formulas, with their propositional variables selected from the set of abducibles.

Conceptually, while we will keep calling the propositional variables involved in our premise abducibles, we should see them from now on simply as a set of independent variables on which the derivation of the sequent’s conclusion from its premise depends.

We define a generator for the set of Mints formulas with help of a DCG grammar that collects its propositional variables from a list of abducibles.

```prolog
mints_formula(P) --> [P].  
mints_formula(~P) --> [P].  
mints_formula((P->Q)) --> [P,Q].  
mints_formula((P->Q)->R) --> [P,Q,R].  
mints_formula((P->(Q->R))) --> [P,Q,R].  
mints_formula((P->(Q v R))) --> [P,Q,R].  
mints_formula((P-> ~Q)) --> [P,Q].  
mints_formula((~P->Q)) --> [P,Q].
```

We extend our DCG, subject to the same limitation on available abducibles, to generate a list of formulas meant to be part of the premise. We will later aggregate this list into a conjunction.

```prolog
mints_conjuncts([],[]).  
mints_conjuncts([F|Fs]) --> mints_formula(F), mints_conjuncts(Fs).
```

Next, we eliminate duplicates with Prolog’s sort/2.

```prolog
mints_conjuncts(A, C) :- mints_conjuncts(Ps, A, []), sort(Ps, C).
```
We derive any_mints_premise/4 in a way similar to any_protasis/6, except that the arguments WithNeg and AggregatorOp become unnecessary and multiple occurrences of any abducible need to be supported. For brevity, we explain our steps as comments in the code.

```
any_mints_premise(Prover,Abducibles,Formula,Premise):-
  abducibles_of(Formula,Abducibles),
  subset_of(Abducibles,Chosen),  % select a subset of Abducibles
  template_from(Abducibles,Atoms), % Atoms is a list of free variables
  part_as_equiv(Atoms,Chosen),  % Chosen provides unique occurrences of Atoms
  mints_conjuncts(Atoms,Conjuncts),  % builds the Mints formulas
  join_with_op((&),Conjuncts,Premise),  % joins Conjuncts into a conjunction
  \+ (call(Prover,Premise->false)),  % ensures Premise is not a contradiction
  call(Prover,Premise->Formula).  % ensure that Premise implies Formula
```

Note that we have limited the length of the premise in the case of the Mints-formulas, arbitrarily, to the number of abducible atoms, that the selection of Atoms has as an upper bound. For more flexibility, this can be lifted to be based on a length parameter passed to any_mints_premise.

### 4.2 Labeling the Variables in the Mints Formulas

We will describe here the implementation of part_as_equiv/2 that will be used to generate variables bound to possibly repeated occurrences of each atom. Note that equalities of logic variables define equivalence classes that correspond to partitions of the set of variables. We implement this simply by selectively unifying them.

The predicate part_as_equiv/2 takes a list of distinct logic variables and generates partition-as-equivalence-relations by unifying them “nondeterministically”. It also collects the unique variables defining the equivalence classes, as a list given by its second argument. It works reversibly, when unique values are given as its second argument and a bound list of free variables as its first.

```
part_as_equiv([],[]).
part_as_equiv([U|Xs], [U|Us]):-complement_of(U,Xs,Rs),part_as_equiv(Rs,Us).
```

To implement it, we split a set repeatedly in subset+complement pairs with help from the predicate complement_of/2.

```
complement_of(_,[],[]).
complement_of(U,[X|Xs],NewZs):-complement_of(U,Xs,Zs),place_element(U,X,Zs,NewZs).
place_element(U,U,Zs,Zs).
place_element(_,X,Zs,[X|Zs]).
```

**Example 7**

Here, we are interested in the reverse use of part_as_equiv, with the list of unique variables as input and a sequence of variables of fixed length but possibly repeated occurrences as output.

```
?- length(Vs,4),part_as_equiv(Vs,[a,b]).
Vs = [a, b, a, a] ; Vs = [a, a, b, a] ; Vs = [a, b, b, a] ; Vs = [a, a, a, b] ; Vs = [a, b, a, b] ; Vs = [a, a, b, b] ; Vs = [a, b, b, b] .
```

We derive weakest_mints_premise/4 in a way similar to weakest_protasis/6 except for passing only the relevant arguments to any_mints_premise/4.
weakest_mints_premise(Prover,Abducibles,Formula,Premise):-
    findall(Premise, any_mints_premise(Prover,Abducibles,Formula,Premise), Premises),
    sort(Premises,Uniques),
    weakest_with(Prover,Uniques,Premise).

Example 8
When using the axiom that conservatively extends IL to the logic of here-and-there (Lifschitz et al. 2001) we observe again that no premise is needed for cprover and that iprover suggests as premises either one of the disjuncts in the axiom, or, more interestingly, $g \rightarrow \neg g$.

?- weakest_mints_premise(cprover,_,(f v (f->g) v ~g),P).
P = true.
?- weakest_mints_premise(iprover,_,(f v (f->g) v ~g),P).
P = f ; P = ~g ; P = ( f->g) ; P = ( g-> ~g).

In fact, we observe:

?- iprover((g -> ~g) <-> ~g).
true.

suggesting that extending IL with:

$f v (f->g) v (g -> \neg g)$

would result in an alternative axiomatization of the logic of here-and-there.

5 Discussion

We hope that the astute reader is aware at this point that the paper is an exploration of the theory behind some fundamental concepts relating abductive reasoning, program synthesis and theorem proving in a concise and easily replicable form, facilitated by the choice of Prolog as our meta-language, but with the possibility of a fairly routine transliteration to a traditional “formulas-on-paper” presentation in mind. As such, the paper can be seen as an executable specification of these concepts. While not neglecting minimal efforts for efficient execution, and some elegance in the coding style, our main priority was to ensure that the paper conveys its message as a fully self-contained literate program.

We have designed our abductive reasoning logic entirely in a proof-theoretical framework, in contrast to the usual model-theoretical semantics, arguably in the original spirit of intuitionistic logic.

Both our weakest protasis-based and weakest Mints formulas-based synthetic assumptions are attempts to recover in IL an analogue of the CNF available for a formula in CL. At the same time, finding the weakest assumptions shares the focus on minimal models encountered in various logic calculi. Our interest in finding weakest assumptions under which the formula becomes a theorem is driven by the transitivity of the partial order induced by $\rightarrow$, given that for a given formula $f$, becoming a theorem under the assumption $w$, ensures also that if $(s\rightarrow w)$, then $(s\rightarrow f)$ is also a theorem, where $w$ denotes a weakest assumption and $s$ denotes a (stronger) assumption that implies it.

We have restricted ourselves to propositional logic but we foresee extensions to stronger logics.
among which Monadic First Order Logic (known as decidable for CL and undecidable for IL),

enhanced with Prolog’s constraint solving mechanisms is a promising option.

While we have forced a clear separation between our meta-language (Prolog) and object-language (IL), it would be quite easy to extend our theorem prover to reflect Prolog’s negation as failure in the object-language as an addition to IL. This would result in a logic with two flavors of negation, similar in the context of IL to the underlying equilibrium logic of ASP.

6 Related work

We refer to (Denecker and Kakas 2002) as still the most lucid and comprehensive overview (also citing 124 papers) on abductive reasoning in Logic Programming and to (Eshghi and Kowalski 1989) as one of the most influential initiators for the interest in the field, with connections explored in depth to negation as failure and non-monotonic reasoning. Some of our examples related to the logic of here-and-there and equilibrium logic (Pearce 1996; Pearce et al. 2000) originate in (Lifschitz et al. 2001), where intermediate logics relevant for the foundation of ASP systems are overviewed. For abductive reasoning in the context of several logics including non-monotonic ones, we mention (Gabbay and Olivetti 2002) and (Gabbay 2000).

By contrast, the novelty of our approach is the generalized view of abductive reasoning as an instance of program synthesis controlled by a theorem prover. Our theorem prover is derived directly from the G4ip sequent calculus (Dyckhoff 1992; Dyckhoff 2016). In (Tarau 2019) details of this derivation process as well as a combinatorial testing framework used to ensure correctness are given. The idea of using a theorem-prover for the synthesis of modal formulas is also present in (Tarau 2020), having as an outcome an embedding of the epistemic logic IEL in IL and a derived theorem prover for that logic. In (Tarau and de Paiva 2020) a theorem prover, restricted to the implicational fragment of IL, is used to derive a theorem prover for implicational linear logic, with help from the Curry-Howard correspondence and the use of linearity of the resulting lambda terms as a filtering mechanism. In this context, the distinct focus of the current paper is on a very general formula synthesis mechanism within IL itself, covering abductive reasoning and emulating in IL key semantic concepts available in CL and intermediate logics.

7 Conclusions

We have presented a fully executable specification of a generalized abductive reasoning framework, that can be relatively easily ported to any logic for which a decision mechanism exists (e.g., as provided by a theorem prover). In particular, this applies to several interesting intermediate logics among which the equilibrium-logic (relevant as a foundation of ASP systems) as well as modal logics and their instantiations as alethic, deontic or epistemic systems. Besides providing (in the form of the concept of weakest protasis) an analogue of the unavailable truth-table models for intuitionistic formulas, we have also generalized our abduced sequent premises to use minimal canonical formulas to which arbitrary IL formulas can be broken down, with the potential of synthesizing salient assumptions that would make a given formula a theorem. When the underlying logic is used to model a set of safety constraints that should always hold, this generalized abduction synthesis could reveal critical missing assumptions, not just as literals but also as a conjunction of interdependencies among them.
References


