# Synthesis of Modality Definitions and a Theorem Prover for Epistemic Intuitionistic Logic

Paul Tarau

Department of Computer Science and Engineering
University of North Texas
*paul.tarau@unt.edu*

**Abstract.** We propose a mechanism for automating discovery of definitions, that, when added to a logic system for which we have a theorem prover, extends it to support an embedding of a new logic system into it. As a result, the synthesized definitions, when added to the prover, implement a prover for the new logic.
As an instance of the proposed mechanism, we derive a Prolog theorem prover for an interesting but unconventional epistemic Logic by starting from the sequent calculus **G4IP** that we extend with operator definitions to obtain an embedding in intuitionistic propositional logic (**IPC**). With help of a candidate definition formula generator, we discover epistemic operators for which axioms and theorems of Artemov and Protopopescu's *Intuitionistic Epistemic Logic* (**IEL**) hold and formulas expected to be non-theorems fail.
We compare the embedding of **IEL** in **IPC** with a similarly discovered successful embedding of Dosen's double negation modality, judged inadequate as an epistemic operator. Finally, we discuss the failure of the *necessitation rule* for an otherwise successful **S4** embedding and share our thoughts about the intuitions explaining these differences between epistemic and alethic modalities in the context of the Brouwer-Heyting-Kolmogorov semantics of intuitionistic reasoning and knowledge acquisition.

**Keywords:** automatic synthesis of logic systems, deriving new theorem provers via program synthesis, epistemic intuitionistic logic, propositional intuitionistic logic, Prolog-based theorem provers, embedding of modal logics into intuitionistic logic.

## 1 Introduction

Deriving new logic systems and discovering relationships between them not only requires a knowledge-intensive understanding of the intricate connections between their axioms and inference rules but it is also a time-intensive trial and error process for the human logician. This is especially the case for logic systems that depart from the usual expectations coming from the prevalent use of classical logic in today's computational tools and methodologies as well from familiarity with more commonly used forms of modal logic (e.g., alethic, temporal).

This motivates our effort to explore ways to automate this process, resulting not only in discovering some salient relationships between new and well-established logic systems, but also in software artifacts (e.g., automated theorem provers) facilitating reasoning in these less explored new logics.

Epistemic Logic Systems have been derived often in parallel and sometime as afterthoughts of alethic Modal Logic Systems, in which modalities are defined by axioms and additional inference rules extending classical logic.

In the context of Answer Set Programming (**ASP**) epistemic logics hosted in this framework like e.g., [1–3] show that *intermediate logics*[1] (e.g., equilibrium logic, [4]) can express epistemic operators by extending them with definitions of epistemic operators.

Steps[2], further below classical logic or **ASP**, are taken in recent work [5], based on the Brouwer-Heyting-Kolmogorov (**BHK**) view of intuitionistic logic that takes into account the constructive nature of knowledge, modeling more accurately the connection between proof systems and the related mental processes. Along these lines, our inquiry into epistemic logic will focus on knowledge vs. truth seen as intuitionistic provability.

Like in the case of embedding epistemic operators into **ASP** systems, but with automation in mind, we will design a synthesis mechanism for epistemic operators via embedding in **IPC**. For this purpose we will generate *candidate formulas* that verify axioms, theorems and rules and fail on expected non-theorems. For this purpose, we will use a lightweight **IPC** *theorem prover* and we will also show that this view generalizes to a mechanism for discovering for a given modal logic, when possible, a simple embedding of the logic into **IPC** and derivation of a theorem prover for it.

Our starting point is Artemov and Protopopescu's *Intuitionistic Epistemic Logic* (**IEL**) [5] that will provide the axioms, theorems and non-theorems stating the requirements that must hold for the definitions extending **IPC**. The discovery mechanism will also bring up Dosen's interpretation of double negation [6] as a potential epistemic operator and we will look into applying the same discovery mechanisms to find an embedding of modal logic **S4** in **IPC**, with special focus on the impact of the *necessitation rule*, which requires that all theorems of the logic are necessarily true.

**The rest of the paper is organized as follows.** Section 2 overviews Artemov and Protopopescu's *Intuitionistic Epistemic Logic* (**IEL**). Section 3 introduces the **G4IP** sequent calculus prover for Intuitionistic Propositional Logic (**IPC**). Section 4 describes the generator for candidate formulas extending **IPC** with modal operator definitions. Section 5 explains the discovering of the definitions that ensure the embedding of **IEL** into **IPC** and the discovering of the embedding of Dosen's double negation as a modality operator. It also discusses the intuitions behind the embedding of **IEL**, including the epistemic equivalent of the necessity rule, in **IPC** and the adequacy of this embedding as a constructive mechanism for reasoning about knowledge. Section 6 studies the case of the **S4** modal logic and the failure of the necessity rule, indicating the difficulty of embedding it in **IPC** by contrast to **IEL**. Section 7 overviews some related work and section 8 concludes the paper.

[1] Logics stronger than intuitionistic but weaker than classical.

[2] Actually infinitely many, as there's an infinite lattice of intermediate logics between classical and intuitionistic logic.

The paper is written as a literate SWI-Prolog program with its extracted code at `https://raw.githubusercontent.com/ptarau/TypesAndProofs/master/ieltp.pro`.

## 2 Overview of Artemov and Protopopescu's IEL logic

In [5] a system for Intuitionistic Epistemic Logic is introduced that

> "maintains the original Brouwer-Heyting-Kolmogorov semantics for intuitionism and is consistent with the well-known approach that intuitionistic knowledge be regarded as the result of verification".

Instead of the classic, alethic-modalities inspired **K** operator for which

$$\mathbf{K}A \to A$$

Artemov and Protopopescu argue that *co-reflection* expresses better the idea of *constructivity of truth*

$$A \to \mathbf{K}A$$

They also argue that this applies to both belief and knowledge i.e., that

> "The verification-based approach allows that justifications more general than proof can be adequate for belief and knowledge".

On the other hand, they consider *intuitionistic reflection* acceptable, expressing the fact that "known propositions cannot be false":

$$\mathbf{K}A \to \neg\neg A$$

Thus, they position intuitionistic knowledge of $A$ between $A$ and $\neg\neg A$ and given that (via Glivenko's transformation [7]) applying double negation to a formula embeds classical propositional calculus into **IPC**, they express this view as:

$$\textit{Intuitionistic Truth} \;\Rightarrow\; \textit{Intuitionistic Knowledge} \;\Rightarrow\; \textit{Classical Truth}.$$

They axiomatize the system **IEL** as follows.

**1**. Axioms of propositional intuitionistic logic;
**2**. $\mathbf{K}(A \to B) \to (\mathbf{K}A \to \mathbf{K}B)$; (distribution)
**3**. $A \to \mathbf{K}A$. (co-reflection)
**4**. $\mathbf{K}A \to \neg\neg A$ (intuitionistic reflection)

**Rule** *Modus Ponens*.

They also argue that a weaker logic of belief ($\mathbf{IEL}^-$) is expressed by considering only axioms **1,2,3**.

## 3 The G4ip prover for IPC

We will describe next our lightweight propositional intuitionistic theorem prover, that will be used to discover an embedding of **IEL** into **IPC**.

### 3.1 The LJT/G4ip calculus, (restricted here to the implicational fragment)

Motivated by problems related to loop avoidance in implementing Gentzen's **LJ** calculus, Roy Dyckhoff [8] introduces the following rules for the **G4ip** calculus[3].

$$LJT_1 : \quad \frac{}{A,\Gamma \vdash A}$$

$$LJT_2 : \quad \frac{A,\Gamma \vdash B}{\Gamma \vdash A \to B}$$

$$LJT_3 : \quad \frac{B,A,\Gamma \vdash G}{A \to B,A,\Gamma \vdash G}$$

$$LJT_4 : \quad \frac{D \to B,\Gamma \vdash C \to D \qquad B,\Gamma \vdash G}{(C \to D) \to B,\Gamma \vdash G}$$

Note that $LJT_4$ ensures termination as formulas in the sequent become smaller in a multiset ordering. The rules work with the context $\Gamma$ being either a multiset or a set, and the calculus is sound and complete for IPC.

For supporting negation, one also needs to add $LJT_5$ that deals with the special term $false$. Then negation of $A$ is defined as $A \to false$.

$$LJT_5 : \quad \frac{}{false,\Gamma \vdash G}$$

Rules for conjunction, disjunction and bi-conditional (not shown here) are also part of the calculus.

As it is not unusual with logic formalisms, the same calculus had been discovered independently in the 1950's by Vorob'ev and in the 80's-90's by Hudelmaier [9, 10].

### 3.2 A Lightweight Theorem Prover for Intuitionistic Propositional Logic

Starting from the sequent calculus for the intuitionistic propositional logic in G4ip [8], to which we have also added rules for the "<->" relation, we obtain the following lightweight **IPC** prover.

```
:- op(525,  fy,  ~ ).
:- op(550, xfy,  & ).    % right associative
:- op(575, xfy,  v ).    % right associative
:- op(600, xfx,  <-> ).  % non associative
```

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

```

[3] Originally called the LJT calculus in [8]. Restricted here to its key implicational fragment.

```
prove_in_ipc(A,Vs):-memberchk(A,Vs),!.
prove_in_ipc(_,Vs):-memberchk(false,Vs),!.
prove_in_ipc(A<->B,Vs):-!,prove_in_ipc(B,[A|Vs]),prove_in_ipc(A,[B|Vs]).
prove_in_ipc((A->B),Vs):-!,prove_in_ipc(B,[A|Vs]).
prove_in_ipc(A & B,Vs):-!,prove_in_ipc(A,Vs),prove_in_ipc(B,Vs).
prove_in_ipc(G,Vs1):- % atomic or disj or false
  select(Red,Vs1,Vs2),
  prove_in_ipc_reduce(Red,G,Vs2,Vs3),
  !,
  prove_in_ipc(G,Vs3).
prove_in_ipc(A v B, Vs):-(prove_in_ipc(A,Vs);prove_in_ipc(B,Vs)),!.
```

```
prove_in_ipc_reduce((A->B),_,Vs1,Vs2):-!,prove_in_ipc_imp(A,B,Vs1,Vs2).
prove_in_ipc_reduce((A & B),_,Vs,[A,B|Vs]):-!.
prove_in_ipc_reduce((A<->B),_,Vs,[(A->B),(B->A)|Vs]):-!.
prove_in_ipc_reduce((A v B),G,Vs,[B|Vs]):-prove_in_ipc(G,[A|Vs]).
```

```
prove_in_ipc_imp((C->D),B,Vs,[B|Vs]):-!,prove_in_ipc((C->D),[(D->B)|Vs]).
prove_in_ipc_imp((C & D),B,Vs,[(C->(D->B))|Vs]):-!.
prove_in_ipc_imp((C v D),B,Vs,[(C->B),(D->B)|Vs]):-!.
prove_in_ipc_imp((C<->D),B,Vs,[((C->D)->((D->C)->B))|Vs]):-!.
prove_in_ipc_imp(A,B,Vs,[B|Vs]):-memberchk(A,Vs).
```

We validate it first by testing it on the implicational subset, derived via the Curry-Howard isomorphism [11], then against Roy Dyckhoff's Prolog implementation[4], working on formulas up to size 12. Finally we run it on human-made tests[5], on which we get no errors, solving correctly 161 problems, with a 60 seconds timeout, compared with the 175 problems solved by Roy Dyckhoff's more refined, heuristics-based 400 lines prover, with the same timeout[6]. We refer to [11] for the derivation steps of variants of this prover working on the implicational and nested Horn clause fragments of **IPC**. While more sophisticated tableau-based provers are available for **IPC** among which we mention the excellent Prolog-based `fCube` [12], our prover's compact size and adequate performance will suffice [7].

## 4 The definition formula generator

We start with a candidate formula generator that we will constrain further to be used for generating candidate definitions of our modal operators.

[4] `https://github.com/ptarau/TypesAndProofs/blob/master/third_party/dyckhoff_orig.pro`

[5] at `http://iltp.de`

[6] `https://github.com/ptarau/TypesAndProofs/blob/master/tester.pro`

[7] In fact, our prover is faster than both fCube and Dyckhoff's prover on the set of formulas of small size on which our definition induction algorithm will run.

## 4.1 Generating Operator Trees

We generate all formulas of a given size by decreasing the available size parameter at each step when nodes are added to a tree representation of a formula. Prolog's **DCG** mechanism is used to collect the leaves of the tree.

```
genOperatorTree(N,Ops,Tree,Leaves):-
  genOperatorTree(Ops,Tree,N,0,Leaves,[]).

genOperatorTree(_,V,N,N)-->[V].
genOperatorTree(Ops,OpAB,SN1,N3)-->
  { SN1>0,N1 is SN1-1,
    member(Op,Ops),make_oper2(Op,A,B,OpAB)
  },
  genOperatorTree(Ops,A,N1,N2),
  genOperatorTree(Ops,B,N2,N3).

make_oper2(Op,A,B,OpAB):-functor(OpAB,Op,2),arg(1,OpAB,A),arg(2,OpAB,B).
```

## 4.2 Synthesizing the definitions of modal operators

As we design a generic definition discovery mechanism, we will denote generically our modal operators as follows.

- "#" for "$\Box$"=necessary and "**K**"=known
- "$*$" for "$\Diamond$"=possible and "**M**"=knowable

After the operator definitions

```
:- op( 500,  fy, #).
:- op( 500,  fy, *).
```

we specify our generator as covering the usual binary operators and we constrain it to have at least one of the leaves of its generated trees to be a variable. Besides the `false` constant used in the definition of negation, we introduce also a new constant symbol "?" assumed not to occur in the language. Its role will be left unspecified until the possible synthesized definitions will be filtered. We will constrain candidate definitions to ensure that axioms and selected theorems hold and selected non-theorems fail.

```
genDef(M,Def):-genDef(M,[(->),(&),(v)],[false,?],Def).

genDef(M,Ops,Cs,(#(X):-T)):-
  between(0,M,N),
  genOperatorTree(N,Ops,T,Vs),
  pick_leaves(Vs,[X|Cs]),
  term_variables(Vs,[X]).
```

Leaves of the generated trees will be picked from a given set.

```
pick_leaves([],_).
pick_leaves([V|Vs],Ls):-member(V,Ls),pick_leaves(Vs,Ls).
```

We first expand our operator definitions for the “`~`” negation and “`*`” modal operator while keeping atomic variables and the special constant `false` untouched.

```
expand_defs(_,false,R) :-!,R=false.
expand_defs(_,A,R) :-atomic(A),!,R= A.
expand_defs(D,~(A),(B->false)) :-!,expand_defs(D,A,B).
expand_defs(D,*(A),R):-!,expand_defs(D,~ (# (~(A))),R).
```

The special case for expanding a candidate operator definition `D` requires a fresh variable for each instance, ensured by Prolog’s built-in `copy_term`.

```
expand_defs(D,#(X),R) :-!,copy_term(D,(#(X):-T)),expand_defs(D,T,R).
```

Other operators are traversed generically by using Prolog’s “`=..`” built-in and by recursing with `expand_def_list` on their arguments.

```
expand_defs(D,A,B) :-
  A=..[F|Xs],
  expand_def_list(D,Xs,Ys),
  B=..[F|Ys].
```

```
expand_def_list(_,[],[]).
expand_def_list(D,[X|Xs],[Y|Ys]) :-
  expand_defs(D,X,Y),
  expand_def_list(D,Xs,Ys).
```

The predicate `prove_with_def` refines our **G4ip** prover by first expanding the definitions extending **IPC** with a given candidate modality.

```
prove_with_def(Def,T0) :-expand_defs(Def,T0,T1),prove_in_ipc(T1,[]).
```

The definition synthesizer will filter the candidate definitions provided by `genDef` such that the predicate `prove_with_def` succeeds on all theorems and fails on all non-theorems, provided as names of the facts of arity 1 containing them.

```
def_synth(M,D):-def_synth(M,iel_th,iel_nth,D).

def_synth(M,Th,NTh,D):-
  genDef(M,D),
  forall(call(Th,T),prove_with_def(D,T)),
  forall(call(NTh,NT), \+prove_with_def(D,NT)).
```

Note that the generator first builds smaller formulas and then larger ones up the specified maximum size.

**Example 1** *Candidate definitions up to size 2*

```
?- forall(genDef(2,Def),println(Def)).
#A :- A
#A :- A -> A
#A :- A -> false
#A :- A -> ?
#A :- false -> A
#A :- ? -> A
```

```
#A :- A & A
#A :- A & false
#A :- A & ?
...
#A :- (A -> ?) -> A
...
#A :- (? v A) v ?
#A :- (? v false) v A
#A :- (? v ?) v A
```

# 5 Discovering the embedding of IEL and Dosen's double negation modality in IPC

We specify a given logic (e.g., **IEL** or **S4**) by stating theorems on which the prover extended with the synthetic definition should succeed and non-theorems on which it should fail.

## 5.1 The discovery mechanism for IEL

We start with the axioms of Artemov and Protopopescu's **IEL** system:

```
iel_th(a -> # a).
iel_th(# (a->b)->(# a-> # b)).
iel_th(# a -> ~ ~ a).
```

Note that the axioms would be enough to specify the logic, but we also add some theorems when intuitively relevant and/or mentioned in [5]. Our Prolog code, running in less than a second, is not slowed down by this in any significant way.

```
iel_th(#   (a & b) <-> (# a & # b)).
iel_th(~ # false).
iel_th(~ (# a & ~ a)).
iel_th(~a -> ~ # a).
iel_th( ~ ~ (# a -> a)).
iel_th(# a & # (a->b) -> # b).
iel_th(* (a & b) <-> (* a & * b)).
iel_th(# a -> * a).
iel_th(# a v # b -> # (a v b) ).
iel_th(# p <-> # # p).
iel_th(* a <-> * * a).
iel_th(a -> *a).
```

Again, following [5], we add our non-theorems.

```
iel_nth(# a -> a).
iel_nth(# (a v b) -> # a v # b).
iel_nth(# a).
iel_nth(~ (# a)).
iel_nth(# false).
iel_nth(# a).
```

```
iel_nth(~ (# a)).
iel_nth(* false).
```

The *necessitation rule* in a modal logic requires that if `T` is a theorem than `#T` is also a theorem. This expresses the fact that the theorems of the logic are *necessarily* true, or in an epistemic context that if `T` is an (intuitionistically proven) theorem, then the agent *knows* `T`. Thus, we define (implicit) facts via a Prolog rule that states that the (generic) necessity operator "`#`" applied to proven theorems or axioms generates new theorems.

```
iel_nec_th(T):-iel_th(T).
iel_nec_th(# T):-iel_th(T).
```

Finally, we obtain the discovery algorithm for **IEL** formula definitions and for **IEL** extended with the necessitation rule.

```
iel_discover:-
  backtrack_over((def_synth(2,iel_th,iel_nth,D),println(D))).

iel_nec_discover:-
  backtrack_over((def_synth(2,iel_nec_th,iel_nth,D),println(D))).

backtrack_over(Goal):-call(Goal),fail;true.

println(T):-numbervars(T,0,_),writeln(T).
```

We run `iel_discover`, ready to see the surviving definition candidates.

**Example 2** *Definition discovery without the necessitation rule.*

```
?- iel_discover.
#A:-(A->false)->A
#A:-(A->false)->false
#A:-(A-> ?)->A
true.
```

**Example 3** *Definition discovery with the necessitation rule.*

```
?- iel_nec_discover.
#A:-(A->false)->A
#A:-(A->false)->false
#A:-(A-> ?)->A
true.
```

Unsurprisingly, the results are the same, as a consequence of axiom `A -> #A`.

Clearly, the formula `#A:-(A->false)->A` is not interesting as it would define knowing something as a contradiction that implies itself.

This brings us to the second definition formula candidate.

### 5.2 Eliminating Dosen's double negation modality

In [2] double negation in IPC is interpreted as a "□" modality. This corresponds to one of the synthetic definitions `#A :- (A->false)->false` that is equivalent in **IPC** to

`#A :- ~~A`. It is argued in [5] that it does not make sense as an epistemic modality, mostly because it would entail that all classical theorems are known intuitionistically.

We eliminate it by requiring the collapsing of "`*`" into "`#`" to be a non-theorem:

```
iel_nth(* a <-> # a).
```

In fact, while *known* (`#`) implies *knowable* (`~#~ = *`), it is reasonable to think, as in most modal logics, that the inverse implication does not hold.

After that, we have:

**Example 4** *The double negation modality is eliminated, as it collapses* `#` *and* `*`.

```
?- iel_discover.
#A:-(A -> ?)->A
true.

?- iel_nec_discover.
#A:-(A -> ?)->A
true.
```

### 5.3 Knowledge as awareness?

This leaves us with the `#A :- (A -> ?) -> A`.

Among the consequences of the fact that intuitionistic provability strictly implies classical, is that there's plenty of room left between `p` and `~~p`, where both `#` and `*` find their place, given that the following implication chain holds.

```
p -> #p -> *p -> ~~p
```

Let us now find an (arguably) intuitive meaning for the "`?`" constant in the definition. The interpretation of knowledge as awareness about truth goes back to [13]. Our final definition of intuitionistic epistemic modality as "`#A :- (A -> ?) -> A`" suggests interpreting "`?`" as awareness of an agent entailed by (a proof of) `A`. With this in mind, one obtains an embedding of **IEL** in **IPC** via the extension

$$\mathbf{K}A \equiv (A \to \mathbf{eureka}) \to A$$

where **eureka** is a new symbol not occurring in the language[8].

In line with the Brouwer-Heyting-Kolmogorov (**BHK**) interpretation of intuitionistic proof, we may say that an agent *knows* `A` *if and only if* `A` *is validated by a proof of* `A` *that induces awareness of the agent about it*.

Thus knowledge of an agent, in this sense, collects facts that are proven constructively in a way that is "understood" by the agent. The consequence

$$\mathbf{K}A \to \neg\neg A$$

would then simply say that intuitionistic truths, that the agent is aware of, are also classically valid.

Thus, we can define our *newly synthesized* prover for **IEL** as follows.

---

[8] Not totally accidentally named, given the way Archimedes expressed his sudden *awareness* about the volume of water displaced by his immersed body.

```
iel_prove(P):-prove_with_def((#A :- (A -> eureka) -> A),P).
```

Interestingly, if one allows `eureka` to occur in the formulas of the language given as input to the prover, then it becomes (the unique) value for which we have equivalence between being known and having a proof.

```
?- iel_prove(#eureka <-> eureka).
true .
```

Similarly, it would also follow that

```
?- iel_prove(*eureka <-> ~ ~ eureka).
true.
```

Thus, one would need to forbid accepting it as part of the prover's language to closely follow the intended semantics of **IEL**.

### 5.4 Discussion

*The most significant consequence of the successful embedding of* **IEL** *into* **IPC** *via the epistemic modality definition* `#A :- (A -> eureka) -> A)` *is that we have actually derived a theorem prover for* **IEL**. The theorem prover is implemented by the predicate `iel_prove/1` by extending a theorem prover for **IPC** with the induced definition.

As the **IPC** fragment with two variables, implication and negation has exactly **518** equivalence classes of formulas [14, 15], one would expect the construction deriving "`*`" from "`#`" to reach a fixpoint. We can use our prover to find out when that happens.

```
?- iel_prove(#p <-> ~ # (~p)).
false.
iel_prove(*p <-> ~(*(~p))).
true.
```

Thus the fixpoint of the construction is "`*`", that we have interpreted as meaning that a proposition is *knowable*. Therefore, the equivalence reads reasonably as "something is knowable if and only if its negation is not knowable". Note also that

```
?- iel_prove(~(*(~p)) -> #p).
false.
```

fails, by contrast to the equivalence $\Box p \equiv \neg\Diamond\neg p$ usual in classical modal logics.

## 6 Discovering an embedding of S4 without the necessitation rule

The fact that both **IPC** and **S4** are known to be PSPACE-complete [16] means that polynomial-time translations exist between them.

In fact, Gödel's translation from **IPC** to **S4** (by prefixing each subformula with the $\Box$ operator) shows that the embedding of **IPC** into **S4** can be achieved quite easily, by using purely syntactic means. However, the (very) few papers attempting the inverse translation [17, 18] rely on methods often involving intricate semantic constructions.

We will use our definition generator to identify the problem that precludes a simple embedding of **S4** into **IPC**.

We start with the axioms of **S4**.

```
s4_th(# a -> a).
s4_th(# (a->b) -> (# a -> # b)).
s4_th(# a -> # # a).
```

We add a few theorems.

```
s4_th(* * a <-> * a).
s4_th(a -> * a).
s4_th(# a -> * a).
s4_th(# a v # b -> # (a v b)).
s4_th(# (a v b) -> # a v # b).
```

We add some non-theorems that ensure additional filtering.

```
s4_nth(# a).
s4_nth(~ (# a)).
s4_nth(# false).
s4_nth(* false).
s4_nth(* a -> # * a). % true only in S5
s4_nth(a -> # a).
s4_nth(* a -> a).
s4_nth(# a <-> ?).
s4_nth(* a <-> ?).
```

Like in the case of **IEL** we define implicit facts stating that the necessitation rule holds.

```
s4_nec_th(T):-s4_th(T).
s4_nec_th(# T):-s4_th(T).
```

Finally we implement the definition discovery predicates and run them.

```
s4_discover:-
  backtrack_over((def_synth(2,s4_th,s4_nth,D),println(D))).

s4_nec_discover:-
  backtrack_over((def_synth(2,s4_nec_th,s4_nth,D),println(D))).
```

**Example 5** *The necessitation rule eliminates all simple embeddings of* **S4** *into* **IPC***, while a lot of definition formulas pass without it.*

```
?- s4_discover.
#A :- A & ?
#A :- ? & A
#A :- A & (A-> ?)
#A :- A & (? -> false)
...
true.

?- s4_nec_discover.
true.
```

Among the definitions succeeding without passing the necessity rule test, one might want to pick `#A :- ? & A` as an approximation of the **S4** “$\Box$” operator. In this case “?”

would simply state that "the IPC prover is sound and complete". Still, given the failure of the necessitation rule, the resulting logic is missing a key aspect of the intended meaning of **S4**-provability.

## 7 Related work

Program synthesis techniques have been around in logic programming with the advent of Inductive Logic Programming [19], but the idea of learning Prolog programs from positive and negative examples goes back to [20]. Our definition synthesizer fits in this paradigm, with focus on the use of a theorem prover of a decidable logic (**IPC**) filtering formulas provided by a definition generator through theorems as positive examples and non-theorems as negative examples. The means we use for our definition synthesis are in fact as simple as those described in [20]. The strength of our approach comes from the use of a theorem prover that efficiently validates or rejects definition candidates. The idea to use the new constant "?" in our synthesizer is inspired by proofs that some fragments of **IPC** reduced to two variables have a (small) finite number of equivalence classes [14, 15] as well as by the introduction of new variables, in work on polynomial embeddings of **S4** into **IPC** [17, 18].

We refer to [5] for a thorough discussion of the merits of **IEL** compared to epistemic logics following closely classical modal logic, but the central idea about using intuitionistic logic is that of *belief and knowledge as the product of verification*. Our embedding of **IEL** in **IPC** can be seen as a simplified view of this process through a generic "awareness of an agent" concept in line with [13].

In [1] the concept of *epistemic specifications* is introduced that support expressing knowledge and belief in an Answer Set Programming framework. Interestingly, refinements of this work like [21] and [3] discuss difficulties related to expressing an assumption like $p \to \mathbf{K}p$ in terms of **ASP-based** epistemic operators.

Equilibrium logic [4] gives a semantics to Answer Set programs by extending the 3-valued intermediate logic of here-and-there **HT** with Nelson's constructive strong negation. In [22] a 5-valued truth-table semantics for equilibrium logic is given. In [23] (and several other papers) epistemic extensions of equilibrium logic [4] are proposed, in which $\mathbf{K}p \to p$. By contrast to "alethic inspired" epistemic logics postulating $\mathbf{K}p \to p$ we closely follow the $p \to \mathbf{K}p$ view on which [5] is centered.

While we have eliminated Dosen's double negation modality [6] as an epistemic operator $\mathbf{K}p \equiv \neg\neg p$, it is significant that it came out as the only other meaningful candidate produced by our definition synthesizer.

This suggests that it might be worth investigating further how a similar definition discovery mechanism as the one we have used for **IEL** and **S4** would work for logics with multiple negation operators like equilibrium logic.

Besides the $\mathbf{K}p \to p$ vs. $p \to \mathbf{K}p$ problem a more general question is the choice of the logic supporting the epistemic operators, among logics with finite truth-value models (e.g., classical logic or equilibrium logic) or, at the limit, intuitionistic logic itself, with no such models. Arguably, this could be application dependent, as epistemic operators built on top of IPC are likely to fit better the landscape with intricate nuances of a richer set of epistemic and doxastic operators, while such operators built on top of finite-valued intermediate logics would benefit from simpler decision procedures and faster evaluation mechanisms.

## 8 Conclusions

We have devised a general mechanism for synthesizing definitions that extend a given logic system endowed with a theorem prover. The set of theorems on which the extended prover should succeed and the set of non-theorems on which it should fail can be seen as a declarative specification of the extended system. Success of the approach on embedding the **IEL** system in **IPC** and failure on trying to embed **S4** has revealed the individual role of the axioms, theorems and rules that specify a given logic system and their interaction with the necessitation rule .

Given its generality, our definition generation technique can be applied also to epistemic or modal logic axiom systems to find out if they have interesting embeddings in **ASP** and superintuitionistic logics for which high quality solvers or theorem provers exist. Our program synthesis process, when the embedding succeeds, provides a way to automate the exploration of a new logic system with help of its derived theorem prover and facilitates the work of the human logician to validate or invalidate the intuitions behind it.

## Acknowledgement

We thank the participants to the **EELP'2019** workshop[9] for their constructive suggestions and comments.

## References

1. Gelfond, M.: Strong Introspection. In: Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 1. AAAI'91, AAAI Press (1991) 386–391
2. Baral, C., Gelfond, G., Son, T.C., Pontelli, E.: Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1. AAMAS '10, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2010) 259–266
3. Shen, Y.D., Eiter, T.: Evaluating epistemic negation in answer set programming. Artif. Intell. **237**(C) (August 2016) 115–135
4. Pearce, D.: A new logical characterisation of stable models and answer sets. In: Selected Papers from the Non-Monotonic Extensions of Logic Programming. NMELP '96, Berlin, Heidelberg, Springer-Verlag (1997) 57–70
5. Artemov, S.N., Protopopescu, T.: Intuitionistic Epistemic Logic. Rew. Symb. Logic **9**(2) (2016) 266–298
6. Dosen, K.: Intuitionistic double negation as a necessity operator. Publications de l'Institut Mathématique, Nouvelle série **35**(49) (1984) 15–20

[9] A forum with no formal proceedings but insightful presentations and lively discussions on epistemic extensions of logic programming systems.

7. Glivenko, V.: Sur la logique de M. Brouwer. Bulletin de la Classe des Sciences **14** (1928) 225–228
8. Dyckhoff, R.: Contraction-free sequent calculi for intuitionistic logic. Journal of Symbolic Logic **57**(3) (1992) 795807
9. Hudelmaier, J.: A PROLOG Program for Intuitionistic Logic. SNS-Bericht-. Universität Tübingen (1988)
10. Hudelmaier, J.: An O(n log n)-Space Decision Procedure for Intuitionistic Propositional Logic. Journal of Logic and Computation **3**(1) (1993) 63–75
11. Tarau, P.: A Combinatorial Testing Framework for Intuitionistic Propositional Theorem Provers. In Alferes, J.J., Johansson, M., eds.: Practical Aspects of Declarative Languages - 21th International Symposium, PADL 2019, Lisbon, Portugal, January 14-15, 2019, Proceedings. Volume 11372 of Lecture Notes in Computer Science., Springer (2019) 115–132
12. Ferrari, M., Fiorentini, C., Fiorino, G.: fcube: An efficient prover for intuitionistic propositional logic. In Fermüller, C.G., Voronkov, A., eds.: Logic for Programming, Artificial Intelligence, and Reasoning, Berlin, Heidelberg, Springer Berlin Heidelberg (2010) 294–301
13. Fagin, R., Halpern, J.Y.: Belief, Awareness, and Limited Reasoning: Preliminary Report. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1. IJCAI'85, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1985) 491–501
14. de Bruijn, N.G.: Exact finite models for minimal propositional calculus over a finite alphabet. Technical Report 75?WSK?02, Technological University Eindhoven (November 1975)
15. Jongh, D.D., Hendriks, L., de Lavalette, G.R.R.: Computations in fragments of intuitionistic propositional logic. J. Autom. Reasoning **7**(4) (1991) 537–561
16. Statman, R.: Intuitionistic Propositional Logic is Polynomial-Space Complete. Theor. Comput. Sci. **9** (1979) 67–72
17. Egly, U.: A Polynomial Translation of Propositional S4 into Propositional Intuitionistic Logic. (2007)
18. Goré, R., Thomson, J.: A Correct Polynomial Translation Of S4 Into Intuitionistic Logic. The Journal of Symbolic Logic **84**(2) (2019) 439–451
19. Muggleton, S.: Inductive logic programming. New Gen. Comput. **8**(4) (February 1991) 295–318
20. Shapiro, E.Y.: An algorithm that infers theories from facts. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1. IJCAI'81, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1981) 446–451
21. Gelfond, M.: New semantics for epistemic specifications. In Delgrande, J.P., Faber, W., eds.: Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings. Volume 6645 of Lecture Notes in Computer Science., Springer (2011) 260–265
22. Kracht, M.: On extensions of intermediate logics by strong negation. Journal of Philosophical Logic **27**(1) (Feb 1998) 49–73
23. del Cerro, L.F., Herzig, A., Su, E.I.: Epistemic Equilibrium Logic. In Yang, Q., Wooldridge, M.J., eds.: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press (2015) 2964–2970