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

Paul Tarau

University of North Texas

LOPSTR'2020

Motivation

- Can we synthesize theorem provers for interesting new logics, automatically?
  - intuitionistic logic formalizes constructive mathematics, and more generally constructive representation and processing of knowledge
  - can an intuitively meaningful epistemic logic be *embedded* into Intuitionistic Propositional Logic (**IPC**)?
  - the interesting new logic: Artemov and Protopopescu's *Intuitionistic Epistemic Logic* (**IEL**), designed to capture the spirit of the original Brouwer-Heyting-Kolmogorov **BHK** semantics of intuitionistic reasoning
  - can we design a mechanism to automate the search for a definition that embeds **IEL** into **IPC**?
  - embedding **S4** modal logic in **IPC** is notoriously hard even if they are both **PSPACE**-complete (contrary to the embedding of **IPC** into **S4**)
  - will the same work for embedding **S4** into **IPC**?
- $\Rightarrow$ our motivation for this work:
  - answering these questions for **IEL** and **S4**
  - design a general methodology for answering similar ones

Overview

- we derive a Prolog theorem prover for **IEL** starting from the sequent calculus **G4IP** that we extend with operator definitions providing 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 (**IEL**) hold and formulas expected to be non-theorems fail $\Rightarrow$ we get our theorem prover for **IEL**!
- we discuss the failure of the *necessitation rule* for an otherwise successful **S4** embedding
- we explain our results in the context of the **BHK** semantics of intuitionistic reasoning and knowledge acquisition
- the paper is a literate Prolog program with its extracted code at `https://raw.githubusercontent.com/ptarau/TypesAndProofs/master/ieltp.pro`.

Artemov and Protopopescu's **IEL** logic

a system for Epistemic Intuitionistic 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

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

- the idea of *constructivity of truth* is better expressed with

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

continued

- *constructivity of truth* 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".

- "known propositions cannot be false":

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

- they position intuitionistic knowledge of A between A and $\neg\neg A$:

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

- applying double negation to a formula embeds classical propositional calculus into **IPC** (Glivenko)

Intuitionistic Truth $\Rightarrow$ *Intuitionistic Knowledge* $\Rightarrow$ *Classical Truth.*

The axioms of **IEL**

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*

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

Why do we need an **IPC** theorem prover?

- can **IEL** be embedded in **IPC**?
- if YES: no new axioms, just definitions for the **IEL** operators
- can we synthesize such definitions?
- yes, generate candidates by increasing order of size
- work on embedding **S4** in **IPC** suggests the use of auxiliary propositional variables as helpers
- use an **IPC** theorem prover to test that, when unfolded, the definitions result in valid formulas
- use axioms and (optionally) theorems as positive examples
- use non-theorems as negative examples
- synthesize the definitions, in the tradition of Inductive Logic Programming

The **LJT/G4ip** calculus (implicational fragment)

Roy Dyckhoff's rules for the **G4ip** (originally called the **LJT**)

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

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

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

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

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

the last rule supports intuitionistic negation

A Lightweight Theorem Prover for Full Intuitionistic Propositional Logic

the LJT/G4ip sequent calculus for the full IPC + rules for "<->":

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

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

continued

```
ljfa_reduce((A->B),_,Vs1,Vs2):-!,ljfa_imp(A,B,Vs1,Vs2).
ljfa_reduce((A & B),_,Vs,[A,B|Vs]):-!.
ljfa_reduce((A<->B),_,Vs,[(A->B),(B->A)|Vs]):-!.
ljfa_reduce((A v B),G,Vs,[B|Vs]):-ljfa(G,[A|Vs]).

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

While tableau-based provers (e.g., **fCube)** are better at handling hard human-made tests, our prover is sound, complete and safe from stack and heap overflows. Also, faster than everything else for small formulas.

The definition formula generator: 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
- after the operator definitions

```
:- op( 500,  fy, #).  % known, interpreted as "necessary" in S4
:- op( 500,  fy, *).  % knowable, interpreted as "possible" in S4
```

  we specify our generator as covering the usual binary operators
- we constrain it to have at least one of the leaves of its generated trees to be a variable
- we add the `false` constant used in the definition of negation
- we introduce a new constant symbol "`?`" assumed not to occur in the language
- we constrain candidate definitions to ensure that axioms and selected theorems hold and selected non-theorems fail
- code in the paper

The definition synthesizer

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

- the definition synthesizer filters the candidate definitions such that the prover `prove_with_def` succeeds on all theorems and fails on all non-theorems

```
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)).
```

- theorems and non-theorems are provided as names of the facts of arity 1 containing them as in:

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

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
```

Discovering the embedding of **IEL** 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
- 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).
```

Some theorems

- the axioms would be enough to specify the logic
- we also add some theorems when intuitively relevant and/or mentioned in the **IEL** paper

```
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).
```

Some non-theorems

- following the **IEL** paper, 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).
```

Adding the necessitation rule

- we also define (implicit) facts for supporting the *necessitation rule* that states that the operator "#" applied to proven theorems or axioms generates new theorems.

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

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

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

Definition discovery for **IEL**

- definition discovery without the necessitation rule

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

- 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 `A -> #A`

Eliminating Dosen's double negation modality

- Dosen interprets double negation in IPC 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 the **IEL** paper that it does not make sense as an epistemic modality because it would entail that all classical theorems are known intuitionistically (given Glivenko's $\neg\neg$ translation)
- we eliminate it by requiring the collapsing of "$\star$" into "#" to be a non-theorem:

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

- while *known* (#) implies *knowable* ($\star$ = ~#~), as in most modal logics, the inverse implication should not hold
- we obtain:

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

Knowledge as awareness?

- thus, our final definition is: `#A :- (A -> ?) -> A`, giving:

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

- what would be an intuitive meaning for the "`?`" constant in the definition?
- Fagin and Halpern 1985: *knowledge as awareness about truth*
- we interpret "`?`" *as awareness of an agent entailed by (a proof of)* `A`
- we obtain 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

- not totally accidentally named **eureka**, given the way Archimedes expressed his sudden *awareness* about the volume of water displaced by his immersed body `:-)`

Knowledge as awareness, continued

- in line with (**BHK**) interpretation of intuitionistic proof, we may say that an agent *knows* `A` **iff** `A` *is validated by a proof of* `A` *that induces awareness of the agent about it*.
- knowledge of an agent, in this sense, collects facts that are proven constructively in a way that is "understood" by the agent
- the consequence

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

  would then simply say that intuitionistic truths, that the agent is aware of, are also classically valid
- finally, we define our prover for **IEL** as follows:

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

Discussion

- the **IPC** fragment with two variables, implication and negation has exactly **518** equivalence classes of formulas
- one would expect the construction deriving "$\star$" from "#" to reach a fixpoint

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

- thus the fixpoint of the construction is "$\star$" that we have interpreted as meaning that a proposition is *knowable*
- *"something is knowable if and only if its negation is not knowable"*

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

- by contrast: the equivalence $\Box p \equiv \neg\Diamond\neg p$ holds in classical modal logics

Discovering an embedding of **S4** (no the necessitation rule)

- axioms

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

- 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).
```

Non-theorems and candidate definitions for **S4**

- some non-theorems

```
s4_nth(# a).
s4_nth(~ (# a)).
s4_nth(# false).
s4_nth(* false).
s4_nth(* a -> # * a). % we do not want S5 !
s4_nth(a -> # a).
s4_nth(* a -> a).
s4_nth(# a <-> ?).
s4_nth(* a <-> ?).
```

- some candidate definitions

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

Failing on the necessitation rule

- 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).
```

- the search procedure:

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

- the necessitation rule eliminates all simple embeddings of **S4** into **IPC**

```
?- s4_nec_discover.
true.
```

Related work

- program synthesis techniques have been around in logic programming with the advent of Inductive Logic Programming but the idea of learning Prolog programs from positive and negative examples goes back to E. Shapiro, in 1981
- 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 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
- also, introduction of new variables is present in work on polynomial embeddings of **S4** into **IPC**

continued

- Pearce's equilibrium logic gives a semantics to **ASP** by extending the 3-valued intermediate logic of here-and-there **HT** with Nelson's constructive strong negation
- Kracht introduces a 5-valued truth-table semantics for equilibrium logic is given, fully describing the two negation operators
- several epistemic extensions of equilibrium logic are proposed, in which $\mathbf{K}p \rightarrow p$
- by contrast to "alethic inspired" epistemic logics postulating $\mathbf{K}p \rightarrow p$ we closely follow the $p \rightarrow \mathbf{K}p$ view on which the **IEL** paper is centered
- 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

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 a 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.
- 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 equilibrium logic and superintuitionistic logics for which high quality solvers or theorem provers exist