Interactive Theorem Proving for Logic and Information

Villadsen, Jørgen; From, Asta Halkjær; Jensen, Alexander Birch; Schlichtkrull, Anders

Published in:
Natural Language Processing in Artificial Intelligence

Link to article, DOI:
10.1007/978-3-030-90138-7_2

Publication date:
2022

Document Version
Peer reviewed version

Interactive Theorem Proving for Logic and Information

Jørgen Villadsen, Asta Halkjær From, Alexander Birch Jensen and
Anders Schlichtkrull

Abstract. Automated reasoning is the study of computer programs that can build proofs of theorems in a logic. Such programs can be either automatic theorem provers or interactive theorem provers. The latter are also called proof assistants because the user constructs the proofs with the help of the system. We focus on the Isabelle proof assistant. The system ensures that the proofs are correct, in contrast to pen-and-paper proofs which must be checked manually. We present applications to logical systems and models of information, in particular selected modal logics extending classical propositional logic. Epistemic logic allows intelligent systems to reason about the knowledge of agents. Public announcements can change the knowledge of the system and its agents. In order to account for this, epistemic logic can be extended to public announcement logic. An axiomatic system consists of axioms and rules of inference for deriving statements in a logic. Sound systems can only derive valid statements and complete systems can derive all valid statements. We describe formalizations of sound and complete axiomatic systems for epistemic logic and public announcement logic, thereby strengthening the foundations of automated reasoning for logic and information.

Keywords: Interactive Theorem Proving, Propositional Logic, Epistemic Logic, Public Announcement Logic, Isabelle/HOL Proof Assistant

Jørgen Villadsen
Technical University of Denmark, Kongens Lyngby, Denmark, e-mail: jovi@dtu.dk

Asta Halkjær From
Technical University of Denmark, Kongens Lyngby, Denmark, e-mail: ahfrom@dtu.dk

Alexander Birch Jensen
Technical University of Denmark, Kongens Lyngby, Denmark, e-mail: aleje@dtu.dk

Anders Schlichtkrull
Aalborg University Copenhagen, Copenhagen, Denmark, e-mail: andsch@cs.aau.dk

1 Introduction

Automated reasoning technology has matured tremendously in the recent decades. However, the main applications are found in verification of hardware and software systems as well as in many areas of mathematics. We present a series of applications to logical systems and models of information, in particular classical propositional logic and selected modal logics extending classical propositional logic.

On the one hand, we interpret interactive theorem proving narrowly and focus on the Isabelle proof assistant [44]. On the other hand, we interpret logic and information broadly and consider three logics in the area: propositional logic, epistemic logic (EL) and public announcement logic (PAL).

Building up to formalizations of formulas, we start with a formalization of binary trees and a number of functions operating on these. Thereafter, we formalize a prover for propositional logic as a simple example to introduce the reader to the idea of formalizing logics in Isabelle. We use a so-called deep embedding of logics where formulas are essentially binary trees. By using a datatype for formulas we can prove soundness, completeness and termination of the prover. Moving on, we formalize epistemic logic, a logic for reasoning about both the factual and higher-order knowledge of agents, and a deductive proof system that enables this reasoning from a few axioms and inference rules. Again we use the deep embedding approach and prove soundness and completeness. Finally, we formalize public announcement logic with countably many agents. Public announcement logic extends epistemic logic with an operator for publicly announcing information. The formalization includes proofs of soundness and completeness for a variant of the well-known PA + DIST! + NEC! axiomatic system. The completeness proof builds on the one of epistemic logic by reducing formulas into that logic.

Our definitions are given in Isabelle's precise language of higher-order logic and every step of our soundness and completeness proofs is mechanically checked. With formalizations of sound and complete axiomatic systems for epistemic logic and public announcement logic, we strengthen the foundations of automated reasoning for logic and information.

The formalizations are available here:

https://hol.compute.dtu.dk/ITPLI

The present paper extends our 3-page paper at the International Workshop on Logical Aspects in Multi-Agent Systems and Strategic Reasoning which was not formally published and covered only the formalization of epistemic logic [19].

Summing up, in the present paper we focus on propositional logic, epistemic logic and public announcement logic. As a supplement to pen-and-paper proofs of soundness and completeness, we describe the use of the powerful Isabelle proof assistant for interactive theorem proving.

Other logics have been formalized in Isabelle. We mention here some of them and leave the rest for our discussion of related work together with results in other proof assistants than Isabelle.

- Michaelis and Nipkow formalize several proof systems for classical propositional logic [38, 39]. From, Eschen and Villadsen formalize a number of axiomatic systems for propositional logic [18]. In the present paper we consider modal logics going beyond classical propositional logic.
- From, Lund and Villadsen formalize a number of small provers for classical propositional logic [20, 70, 71]. In the present paper we use a similar prover as a motivational example.

We recommend the survey on the use of formalizations in computer science by Ringer et al. [58] and the state-of-the-art in mathematics in form of the official published account of the now completed Flyspeck project [24].

The paper is organized as follows: Sect. 2 introduces the reader to Isabelle/HOL and how to deeply embed logics. Sect. 3 explains our formalization of epistemic logic. Sect. 4 explains our formalization of public announcement logic. We discuss related work in Sect. 5 and conclude in Sect. 6.

2 Isabelle/HOL and Deep Embeddings of Logics

Isabelle is a generic proof assistant originally developed at the University of Cambridge and Technische Universität München [44]. The most used instance of Isabelle today is Isabelle/HOL, based on classical higher-order logic, and in the following we often use the name Isabelle to refer to Isabelle/HOL.

In order to provide a gentle introduction to programming and proving in Isabelle, we start with a formalization of binary trees and a number of functions operating on these. We further prove a few interesting properties about these functions. In Isabelle/HOL, programming is not limited to the computable fragments of HOL. For instance, a function may return a boolean value that is the result of quantifying over all elements of a type, e.g. stating that all natural numbers are either odd or even. As such, the concept of programming in Isabelle/HOL goes beyond its usual meaning in the context of traditional programming languages like Haskell and Java.

Finally, we briefly consider a formalization of a prover for propositional logic. This mainly serves the purpose of introducing the reader to formalizing logics in Isabelle using a deep embedding approach. In this approach, formulas are defined as a datatype which enables the definition of semantics, a proof system and a small prover as functions that operate on this datatype. In turn, we can prove termination, soundness and completeness of the prover.

2.1 Formally Verified Functional Programming

The following is a rather straightforward example of formally verified functional programming in Isabelle/HOL: a typical solution to an exercise in the Isabelle tutorial [43]. We start with a datatype of trees with labels at the nodes. The labels

can be of any type, as specified by the type variable $'a$, and so-called cartouches delineate the three components of a *Node*:

```
datatype 'a tree = Tip | Node ⟨'a tree⟩ ⟨'a⟩ ⟨'a tree⟩
```

We may collect the contents of such trees into a set by writing a simple functional program:

```
fun set :: ⟨'a tree ⇒ 'a set⟩ where
  ⟨set Tip = {}⟩ |
  ⟨set (Node l a r) = set l ∪ {a} ∪ set r⟩
```

Note that we can use the usual set notation and operators in our definition. The type declaration can be omitted in which case it is inferred automatically.

We can then write a predicate on trees labelled by integers that checks if they are binary search trees:

```
fun ord :: ⟨int tree ⇒ bool⟩ where
  ⟨ord Tip = True⟩ |
  ⟨ord (Node l a r) = ((∀i ∈ set l. i < a) ∧ ord l ∧ (∀i ∈ set r. a < i) ∧ ord r)⟩
```

This checks if they are ordered such that for all nodes, every element in the left subtree is smaller than the element at the node while every element in the right subtree is larger. The following insertion function is supposed to preserve this order:

```
fun ins :: ⟨int ⇒ int tree ⇒ int tree⟩ where
  ⟨ins i Tip = Node Tip i Tip⟩ |
  ⟨ins i (Node l a r) = (if i < a then Node (ins i l) a r else
                         if a < i then Node l a (ins i r) else Node l a r)⟩
```

In the *ord* function we have exploited universal quantification over a finite set, which is computable, but really this program could also be written in an ordinary functional language. This is a good thing as it helps build familiarity with the proof assistant. The next two lines take things a step further:

```
theorem [simp]: ⟨set (ins i t) = {i} ∪ set t⟩
  by (induct t) auto
```

There could potentially be a mistake in the *ins* function where certain elements were not inserted or other elements forgotten. Moreover, we might have to test a lot of inputs to uncover such a mistake. The theorem above, stated for all elements and all trees, rules out such errors. The proof works by induction on the tree and using Isabelle's proof method *auto* to discharge the two resulting cases.

With that result in hand we can also prove that *ins* preserves the binary search tree order:

```
theorem ⟨ord t ⟹ ord (ins i t)⟩
  by (induct t) simp-all
```

Writing a machine-checked proof requires a higher level of abstraction, considering both how properties are expressed and proved.

2.2 Termination

So far we have only considered programs that are trivially total. The **fun** command will prove both pattern completeness and termination automatically. An advanced alternative is to use the **function** command, which does not prove either, and thus we have to do so manually afterwards, for example using Isar for the formal proofs [73].

Pattern completeness must be proved immediately, here with *simp-all*, and termination is shown later with the **termination** command.

We need to prove the termination of our micro provers manually. To illustrate the technique, we consider the McCarthy 91 function, which is an old test case for formal verification [32, 35]. The definition itself is simple, but the nested recursion makes termination non-obvious:

```
function M :: ⟨int ⇒ int⟩ where ⟨M i = (if 100 < i then i − 10 else M (M (i + 11)))⟩
 by simp-all
```

It is called the 91 function because $M\ i = 91$ for all $i \leq 100$ (and $M\ i = i - 10$ otherwise). This is easy to show once termination has been established. We do so below.

To prove termination we show a well-founded relation between the recursive calls and function input:

```
termination
proof
 let ?R = ⟨measure (λi. nat (101 − i))⟩
 show ⟨wf ?R⟩
  by simp
```

Briefly, $(x, y) \in measure f \longleftrightarrow fx < fy$. Any relation defined via *measure* is well-founded by construction. What remains to be shown is that both $i + 11$ and $M(i + 11)$ are related to i, to justify the inner and outer recursive call, respectively. We consider only the branch of the *if* where the recursion happens and as such the first case is trivial given our measure:

```
 fix i :: int
 assume ∗: ⟨¬ 100 < i⟩
 then show ⟨(i + 11, i) ∈ ?R⟩
  by simp
```

For the other case, we assume that $i+11$ is an input that M terminates for, as expressed by *M-dom*:

```
 assume ⟨M-dom (i + 11)⟩
```

This *M-dom* predicate allows us to prove properties about the input that M terminates on, even though we are still to prove that this is in fact all input. In particular, we note that when M terminates, the output is "mostly" larger than the input:

```
 moreover have ⟨M-dom j ⟹ j − 11 < M j⟩ for j
  by (induct j rule: M.pinduct) (auto simp: M.psimps)
```

Since the inner recursive call is on $i + 11$, the output is in fact larger than the input i and this is enough to relate the two, proving termination of the outer recursive call:

```
  ultimately have ‹i + 11 − 11 < M (i + 11)›
    by blast
  then show ‹(M (i + 11), i) ∈ ?R›
    using ∗ by simp
qed
```

Having proved termination, we can now perform induction over the call graph (as expressed by *M.induct*) to prove that the function can be defined without recursion:

```
theorem ‹M i = (if 100 < i then i − 10 else 91)›
  by (induct i rule: M.induct) simp
```

This was an example of a function with a difficult termination proof. We also need to give explicit measures to prove termination of our provers in the coming sections but then the automation takes over, making them more suitable as starting points for exploration. Coming up with the measure can be tricky enough without struggling to prove that it works. We note that this declarative way of proving termination is similar to how a mathematician would do it.

2.3 A Prover for Propositional Logic

The following is a formalization of a simple prover for propositional logic. The prover is implicitly based on a sequent calculus for formulas in negation normal form.

We start with a datatype for formulas:

```
datatype 'a form = Paf 'a | Naf 'a | Con ‹'a form› ‹'a form› | Dis ‹'a form› ‹'a form›
```

Formulas can be combined using conjunction (*Con*) and disjunction (*Dis*). The type variable $'a$ allows for any representation of atomic formulas. We do not include negation as usual; instead, an atomic formula can appear as either positive (*Paf*: positive atomic formula) or negative (*Naf*: negative atomic formula).

The following function defines the semantics of formulas, where an interpretation i maps elements of the type $'a$ to truth values:

```
fun val where
  ‹val i (Paf n) = i n› |
  ‹val i (Naf n) = (¬ i n)› |
  ‹val i (Con p q) = (val i p ∧ val i q)› |
  ‹val i (Dis p q) = (val i p ∨ val i q)›
```

We exploit the built-in Boolean operators for negation, conjunction and disjunction.

Alongside the semantics, we define a sequent calculus as a function for proving formulas:

function *cal* **where**
‹*cal e* [] = (∃ *n* ∈ *fst e*. *n* ∈ *snd e*)› |
‹*cal e* (*Paf n* # *s*) = *cal* ({*n*} ∪ *fst e*, *snd e*) *s*› |
‹*cal e* (*Naf n* # *s*) = *cal* (*fst e*, *snd e* ∪ {*n*}) *s*› |
‹*cal e* (*Con p q* # *s*) = (*cal e* (*p* # *s*) ∧ *cal e* (*q* # *s*))› |
‹*cal e* (*Dis p q* # *s*) = *cal e* (*p* # *q* # *s*)›
by *pat-completeness simp-all*

The sequent calculus operates on a list of formulas, recursively decomposing them. We construct a set of the positive and a set of the negative literals in *e*. The function terminates once the list of formulas is empty — the truth is determined by whether some atom appears in both literal sets.

We need to prove that our *cal* function terminates:

termination by (*relation* ‹*measure* (λ(-, *s*). ∑ *p* ← *s*. *size p*)›) *simp-all*

We obtain a termination proof by providing a suitable measure based on the second argument of the *cal* function: the sum of sizes of the formulas in the list that we decompose.

Because we have defined our sequent calculus as a function, we can immediately obtain a prover by proper initialization of this function:

definition ‹*prover p* ≡ *cal* ({}, {}) [*p*]›

We showcase the prover by running it on a list of formulas (applied to each element individually):

value ‹*map prover* [*Paf n*, *Naf n*, *Con* (*Paf n*) (*Naf n*), *Dis* (*Paf n*) (*Naf n*)]›

Trivially, only the last formula is a tautology so the result is a list with three *False* values and then a single *True* value. Isabelle interactively displays the result of running the prover on the formulas.

We now move on to the question of soundness and completeness for the sequent calculus. We first define an intermediate abbreviation *sat* that captures that at least one literal in *e* (positive or negative) is satisfied by the interpretation *i*.

abbreviation ‹*sat i e* ≡ (∃ *n* ∈ *fst e*. *i n*) ∨ (∃ *n* ∈ *snd e*. ¬ *i n*)›

This definition is useful for stating the soundness and completeness properties of our sequent calculus:

lemma *sound-and-complete*: ‹*cal e s* ⟷ (∀ *i*. (∃ *p* ∈ *set s*. *val i p*) ∨ *sat i e*)›
by (*induct rule*: *cal*.*induct*) *auto*

Because we state soundness and completeness as a single property, and for any call pattern of *cal*, we need to consider both the contents of the sets of positive and negative literals *e*, and the list of formulas *s*. The sequent calculus returns true if and only if, for all interpretations, truth either follows from a formula in the list or from one of the literals. The proof is by induction over the rules of the sequent calculus.

We finally formulate soundness and completeness for the prover:

```
theorem main: ⟨prover p ⟷ (∀i. val i p)⟩
  unfolding sound-and-complete prover-def by simp
```

The stated lemma is weaker than for the sequent calculus and a proof can be obtained by simple rewriting. As such, the proof goal is easily discharged by Isabelle's automation.

3 Epistemic Logic

Epistemic logic provides a foundation for reasoning about the knowledge of agents, both factual ("I know the sky is blue") and higher-order ("I know that you know that I know the sky is blue"). A deductive proof system enables this reasoning with just a few axioms and inference rules. We formalize epistemic logic with countably many agents in the proof assistant Isabelle/HOL [16]. We include soundness and completeness proofs for the axiomatic system K_n based on the textbook *Reasoning About Knowledge* by Fagin, Halpern, Moses and Vardi [14]. Our definitions and proofs are specified in the precise language of higher-order logic and every step of our reasoning is mechanically checked. While the results are not new, we use them to showcase the level of precision and guarantee achievable by formalizing work in a proof assistant. Our formalization can also serve as starting point for similar logics or proof systems.

Our completeness proof does not follow the one by Fagin et al. [14] to the letter but is inspired by Fitting's [15] consistency properties as formalized by Berghofer [5]. We have adapted them from first-order logic to epistemic logic.

3.1 Syntax and Semantics

The formal language $\mathcal{L}$ for epistemic logic is a propositional language extended with modal operators $K_1, \ldots, K_n$ for expressing knowledge of agents, for example the formula

$$K_1\varphi \wedge K_2K_1\varphi \wedge \neg K_1K_2K_1\varphi$$

states that: (i) agent 1 knows φ, (ii) agent 2 knows that agent 1 knows φ, but (iii) agent 1 does not know that agent 2 knows (i).

The language is deeply embedded as a datatype in Isabelle/HOL:

```
datatype 'i fm
  = FF (⊥)
  | Pro id
  | Dis ⟨'i fm⟩ ⟨'i fm⟩ (infixr ∨ 30)
  | Con ⟨'i fm⟩ ⟨'i fm⟩ (infixr ∧ 35)
  | Imp ⟨'i fm⟩ ⟨'i fm⟩ (infixr ⟶ 25)
  | K 'i ⟨'i fm⟩
```

We define a constructor for each primitive of our syntax, e.g. *FF* for falsity with the alternative notation ⊥. Similarly, we give infix syntax for the binary connectives, which all associate to the right and are given suitable precedences.

The type *id* is an abbreviation for strings of characters, used as labels for the propositions. We fix this instead of using a type variable in order to ease notation later.

The type variable $'i$ is an arbitrary type for agents. In our informal example, we used natural numbers, but we do not commit ourselves to any specific type. Our soundness proof holds for any type while the completeness proof holds for any countable type $'i$. We need the agent labels $'i$ to be countable, such that the language itself is countable. Countability of the syntax is a standard prerequisite for our way of proving completeness.

We introduce negation into the syntax as an abbreviation:

abbreviation *Neg* (¬ - [*40*] *40*) **where**
‹*Neg p* ≡ *p* ⟶ ⊥›

The semantics of epistemic logic formulas is based on a model of possible worlds as formalized by Kripke structures:

datatype ('*i*, '*s*) *kripke* = *Kripke* (π: ‹'*s* ⇒ *id* ⇒ *bool*›) (𝒦: ‹'*i* ⇒ '*s* ⇒ '*s set*›)

There are two components: an interpretation π that assigns truth values to propositions for each state (possible world), and a relation $\mathcal{K}$ that, when viewed as a function, takes an agent and a state and returns a set of states. This set is to be understood as the states the agent considers possible given the information available in the input state. We should mention the type variables $('i,\ 's)$. The type $'i$ is again an arbitrary type for agents while $'s$ is the type of states. Thus, the formalization is generic over the type of agents and possible worlds.

The double turnstile, $M,\ s \models \varphi$, denotes the semantics of a formula $\varphi \in \mathcal{L}$ under a Kripke structure M and state s. We formalize it as the following function:

primrec *semantics* :: ‹('*i*, '*s*) *kripke* ⇒ '*s* ⇒ '*i fm* ⇒ *bool*›
(-, - ⊨ - [*50*,*50*] *50*) **where**
‹(-, - ⊨ ⊥) = *False*›
| ‹(*M*, *s* ⊨ *Pro i*) = π *M s i*›
| ‹(*M*, *s* ⊨ (*p* ∨ *q*)) = ((*M*, *s* ⊨ *p*) ∨ (*M*, *s* ⊨ *q*))›
| ‹(*M*, *s* ⊨ (*p* ∧ *q*)) = ((*M*, *s* ⊨ *p*) ∧ (*M*, *s* ⊨ *q*))›
| ‹(*M*, *s* ⊨ (*p* ⟶ *q*)) = ((*M*, *s* ⊨ *p*) ⟶ (*M*, *s* ⊨ *q*))›
| ‹(*M*, *s* ⊨ *K i p*) = (∀*t* ∈ 𝒦 *M i s*. *M*, *t* ⊨ *p*)›

No combination of model and state satisfies ⊥. The logical operators are defined by recursively obtaining the semantics of each subformula and combining the Boolean values through the built-in operators in Isabelle/HOL. The case for a proposition i looks up and returns the truth value of s and i in $\pi\ M$ (the latter gives the π component of the Kripke structure M). Lastly, we have the case for a modal operator $K_i\ p$ which requires the semantics of p to be true in every state agent i considers possible (from the current state).

With the semantics in place, we can prove various properties of the modal operator K_i, say, (see the formalization for the proof):

$$\text{A1}\ \frac{p \text{ is a propositional tautology}}{\vdash p} \qquad \text{A2}\ \frac{}{\vdash K\ i\ p \wedge K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ q}$$

$$\text{R1}\ \frac{\vdash p \qquad \vdash p \longrightarrow q}{\vdash q} \qquad \text{R2}\ \frac{\vdash p}{\vdash K\ i\ p}$$

Fig. 1 Our axiomatic system for epistemic logic.

theorem *distribution*: ‹$M, s \models (K\ i\ p \wedge K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ q)$›

The above states that the operator K_i distributes over implication.

3.2 Axiomatic System

The distribution theorem can be recognized in the very compact axiomatic system K_n (cf. Fig. 1). We adopt the usual syntax that the provability of a formula $\varphi \in \mathcal{L}$ is denoted by the turnstile symbol: $\vdash \varphi$. The system is inductively defined as follows:

inductive *SystemK* :: ‹*'i fm* ⇒ *bool*› (⊢ - [*50*] *50*) **where**
A1: ‹*tautology p* ⟹ ⊢ *p*›
| *A2*: ‹⊢ (*K i p* ∧ *K i* (*p* ⟶ *q*) ⟶ *K i q*)›
| *R1*: ‹⊢ *p* ⟹ ⊢ (*p* ⟶ *q*) ⟹ ⊢ *q*›
| *R2*: ‹⊢ *p* ⟹ ⊢ *K i p*›

$A1$ states that any classical propositional tautology is provable, $A2$ is similar to the distribution theorem, $R1$ is simply modus ponens and $R2$ states that agents also know the provable formulas. The definition *tautology* in $A1$ relies on a semantics that treats modal formulas $K_i\varphi$ as if they were propositional symbols. This is the semantic equivalent of allowing all substitution instances of propositional tautologies, but is simpler to formalize.

3.3 Soundness

For the axiomatic system K to be sound, every formula in $\mathcal{L}$ provable in system K_n must be valid with respect to the semantics:

$$\forall \varphi \in \mathcal{L}.\ \vdash \varphi \longrightarrow (\forall M, s.\ M, s \models \varphi)$$

That is, no combination of proof rules leads to a formula that is not valid. It does not follow that all valid formulas are provable, however, which is why we also need completeness.

Our formalized proof of soundness requires extra work for the rule $A1$. The following theorem states soundness for this rule:

theorem *tautology*: ‹ *tautology* $p \Longrightarrow M, s \models p$ ›

Note that the quantification $p \in \mathcal{L}$ and $\forall M\ s$ is implicit in Isabelle/HOL. See the formalization for the proof.

Proving soundness for system K_n is now straightforward. The following theorem captures the property:

theorem *soundness*: ‹ $\vdash p \Longrightarrow M, s \models p$ ›
by (*induct p arbitrary*: *s rule*: *SystemK.induct*) (*simp-all add*: *tautology*)

The proof strategy is to apply induction over the rules of the system. Once we supply the *tautology* theorem, the simplification proof method in Isabelle/HOL discharges all subgoals.

3.4 Completeness

We now want to demonstrate that system K_n is not only sound, but also complete, namely that every valid formula in $\mathcal{L}$ is provable:

$$\forall \varphi \in \mathcal{L}.\ (\forall M,\ s.\ M,\ s \models \varphi) \longrightarrow \vdash \varphi$$

The formalized proof follows Fagin et al. [14] and builds on maximal consistent sets of formulas. A formula φ is K_n-consistent if its negation is not provable: $\nvdash \neg\varphi$. A finite set of formulas $\varphi_1, \ldots, \varphi_n$ is K_n-consistent if we cannot prove that they imply a contradiction: $\nvdash \varphi_1 \longrightarrow \ldots \longrightarrow \varphi_n \longrightarrow \bot$. Finally, an infinite set of formulas is K_n-consistent if all its finite subsets are.

Instead of working directly with this definition, we start from Fitting's consistency properties [5], which define the class C of consistent sets S directly from the connectives of the formula, instead of referencing the axiom system:

definition *consistency* :: ‹ *'i fm set set* ⇒ *bool* › **where**
‹ *consistency* $C \equiv \forall S \in C.$
$(\forall p.\ \neg\ (Pro\ p \in S \wedge (\neg\ Pro\ p) \in S)) \wedge$
$\bot \notin S \wedge$
$(\forall Z.\ (\neg\ (\neg\ Z)) \in S \longrightarrow S \cup \{Z\} \in C) \wedge$
$(\forall A\ B.\ (A \wedge B) \in S \longrightarrow S \cup \{A, B\} \in C) \wedge$
$(\forall A\ B.\ (\neg\ (A \vee B)) \in S \longrightarrow S \cup \{\neg A, \neg B\} \in C) \wedge$
$(\forall A\ B.\ (A \vee B) \in S \longrightarrow S \cup \{A\} \in C \vee S \cup \{B\} \in C) \wedge$
$(\forall A\ B.\ (\neg\ (A \wedge B)) \in S \longrightarrow S \cup \{\neg A\} \in C \vee S \cup \{\neg B\} \in C) \wedge$
$(\forall A\ B.\ (A \longrightarrow B) \in S \longrightarrow S \cup \{\neg A\} \in C \vee S \cup \{B\} \in C) \wedge$
$(\forall A\ B.\ (\neg\ (A \longrightarrow B)) \in S \longrightarrow S \cup \{A, \neg B\} \in C) \wedge$
$(\forall A.\ tautology\ A \longrightarrow S \cup \{A\} \in C) \wedge$
$(\forall A\ i.\ \neg\ (K\ i\ A \in S \wedge (\neg\ K\ i\ A) \in S))$ ›

All but the last two conditions are standard and ensure downwards saturation [66] of each set: the satisfiability of any member is guaranteed by conditions on its

subformulas, and consistency is ensured at the bottom. The penultimate line ensures that the consistent sets contain all tautologies. This is a technical trick that makes them easier to work with: since any tautology cannot break consistency, we might as well include them. Similarly, the last condition ensures that no agent both knows and does not know the same formula A.

We connect the definition of consistency to provability in system K_n through the following theorem:

theorem *K-consistency*: ‹*consistency* {*set G* | *G*. ¬ ⊢ *imply G* ⊥}›

The completeness proof follows the usual recipe: (i) assume a valid formula φ has no derivation (ii) then its negation is K_n-consistent and (iii) we can extend the set $\{\neg\varphi\}$ in a standard way (due to Lindenbaum [67]) to a maximally consistent set [14] which (iv) has a model. This contradicts the validity assumption. The completeness theorem is:

theorem *completeness*:
 assumes ‹∀ (*M* :: (*'i* :: *countable*, *'i fm set*) *kripke*) *s*. *M*, *s* ⊨ *p*›
 shows ‹⊢ *p*›

For technical reasons we have to require validity in a specific universe, namely in which the possible worlds are sets of formulas, but this is implied by the usual assumption of validity in all universes. Given the provability of p, that is $\vdash p$, the soundness results implies that p is valid in all universes.

4 Public Announcement Logic

We now move beyond static knowledge of agents and consider information updates as well. The formal language $\mathcal{L}_!$ for public announcement logic is an extension of that of epistemic logic with the operator $[r]_!\ p$ for any formulas r and p meaning "p is true after the public announcement of r". For example, $[K_1\rho \wedge_!\ K_2\sigma]_!\ \tau$ means that τ is true after the public announcement that agent 1 knows ρ and agent 2 knows σ. In the formalization [17], we again deeply embed the language as a datatype in Isabelle/HOL:

datatype *'i pfm*
 = *FF* (⊥!)
 | *Pro' id* (*Pro*!)
 | *Dis* ‹*'i pfm*› ‹*'i pfm*› (**infixr** ∨! *30*)
 | *Con* ‹*'i pfm*› ‹*'i pfm*› (**infixr** ∧! *35*)
 | *Imp* ‹*'i pfm*› ‹*'i pfm*› (**infixr** ⟶! *25*)
 | *K' 'i* ‹*'i pfm*› (*K*!)
 | *Ann* ‹*'i pfm*› ‹*'i pfm*› ([-]! - [*50*, *50*] *50*)

We have added primes to some constructors to disambiguate them from the epistemic logic. We say that a formula is *static* if it does not contain any announcement operators.

$$\text{PA1}\ \frac{p \text{ is a propositional tautology}}{\vdash_! p}$$

$$\text{PA2}\ \frac{}{\vdash_! K_! \ i\ p \wedge_! K_! \ i\ (p \longrightarrow_! q) \longrightarrow_! K_! \ i\ q}$$

$$\text{PR1}\ \frac{\vdash_! p \qquad \vdash_! p \longrightarrow_! q}{\vdash_! q} \qquad \text{PR2}\ \frac{\vdash_! p}{\vdash_! K_! \ i\ p} \qquad \text{PR3}\ \frac{\vdash_! p}{\vdash_! [r]_! \ p}$$

$$\text{PFF}\ \frac{}{\vdash_! ([r]_! \ \bot_! \longleftrightarrow_! (r \longrightarrow_! \bot_!))}$$

$$\text{PPro}\ \frac{}{\vdash_! [r]_! \ x \longleftrightarrow_! (r \longleftrightarrow_! x)}$$

$$\text{PDis}\ \frac{}{\vdash_! ([r]_! \ (p \vee_! q) \longleftrightarrow_! [r]_! \ p \vee_! [r]_! \ q)}$$

$$\text{PCon}\ \frac{}{\vdash_! ([r]_! \ (p \wedge_! q) \longleftrightarrow_! [r]_! \ p \wedge_! [r]_! \ q)}$$

$$\text{PImp}\ \frac{}{\vdash_! ([r]_! \ (p \longrightarrow_! q) \longleftrightarrow_! [r]_! \ p \longrightarrow_! [r]_! \ q)}$$

$$\text{PK}\ \frac{}{\vdash_! (([r]_! \ K_! \ ip) \longleftrightarrow_! r \longrightarrow_! K_! \ i([r]_! \ p)))}$$

Fig. 2 Our axiomatic system for public announcement logic.

The bi-implication operator is central to our development and we introduce it as an abbreviation:

abbreviation *PIff* :: ⟨ *'i pfm ⇒ 'i pfm ⇒ 'i pfm* ⟩ (**infixr** $\longleftrightarrow_!$ *25*) **where**
⟨ $p \longleftrightarrow_! q \equiv (p \longrightarrow_! q) \wedge_! (q \longrightarrow_! p)$ ⟩

The semantics depend on the notion of the restriction of a model to the worlds in which a specific formula is true. We formalize the semantics as the function *psemantics* and restriction as the function *restrict*. They are defined by mutual recursion:

fun
psemantics :: ⟨ (*'i*, *'w*) *kripke ⇒ 'w ⇒ 'i pfm ⇒ bool* ⟩ (-, - $\models_!$ - [*50*, *50*] *50*) **and**
restrict :: ⟨ (*'i*, *'w*) *kripke ⇒ 'i pfm ⇒* (*'i*, *'w*) *kripke* ⟩ **where**
⟨ $(M, w \models_! \bot_!) = False$ ⟩
| ⟨ $(M, w \models_! Pro_! \ x) = \pi\ M\ w\ x$ ⟩
| ⟨ $(M, w \models_! (p \vee_! q)) = ((M, w \models_! p) \vee (M, w \models_! q))$ ⟩
| ⟨ $(M, w \models_! (p \wedge_! q)) = ((M, w \models_! p) \wedge (M, w \models_! q))$ ⟩
| ⟨ $(M, w \models_! (p \longrightarrow_! q)) = ((M, w \models_! p) \longrightarrow (M, w \models_! q))$ ⟩
| ⟨ $(M, w \models_! K_! \ i\ p) = (\forall v \in \mathcal{K}\ M\ i\ w.\ M, v \models_! p)$ ⟩
| ⟨ $(M, w \models_! [r]_! \ p) = ((M, w \models_! r) \longrightarrow (restrict\ M\ r, w \models_! p))$ ⟩
| ⟨ $restrict\ M\ p = Kripke\ (\pi\ M)\ (\lambda i\ w.\ \{v.\ v \in \mathcal{K}\ M\ i\ w \wedge (M, v \models_! p)\})$ ⟩

As can be seen, the semantics for each formula is defined the same as for epistemic logic, a semantics for $[\_]_!$ is added, and *restrict* is defined.

We restrict the model, not by removing worlds but by removing every agent's accessibility to those worlds. The idea for that semantics is that for $[r]_!\ p$ to be true in model M and world w, either p is falsified at M and w, a false announcement, or p is satisfied at w in the restricted world *restrict M* where only p-worlds are accessible.

4.1 Axiomatic System

We adapt the syntax $\vdash_!\ \rho$ for the provability of ρ in the following axiomatic system inspired by the system described by Baltag and Renne [2]. It is defined inductively (cf. Fig. 2):

inductive *PA* :: ⟨ *'i pfm* ⇒ *bool* ⟩ (⊢! - [*50*] *50*) **where**
PA1: ⟨ *ptautology p* ⟹ ⊢! *p* ⟩
| *PA2*: ⟨ ⊢! (*K*! *i p* ∧! *K*! *i* (*p* ⟶! *q*) ⟶! *K*! *i q*) ⟩
| *PR1*: ⟨ ⊢! *p* ⟹ ⊢! (*p* ⟶! *q*) ⟹ ⊢! *q* ⟩
| *PR2*: ⟨ ⊢! *p* ⟹ ⊢! *K*! *i p* ⟩
| *PR3*: ⟨ ⊢! *p* ⟹ ⊢! [*r*]! *p* ⟩
| *PFF*: ⟨ ⊢! ([*r*]! ⊥! ⟷! (*r* ⟶! ⊥!)) ⟩
| *PPro*: ⟨ ⊢! ([*r*]! *Pro*! *x* ⟷! (*r* ⟶! *Pro*! *x*)) ⟩
| *PDis*: ⟨ ⊢! ([*r*]! (*p* ∨! *q*) ⟷! [*r*]! *p* ∨! [*r*]! *q*) ⟩
| *PCon*: ⟨ ⊢! ([*r*]! (*p* ∧! *q*) ⟷! [*r*]! *p* ∧! [*r*]! *q*) ⟩
| *PImp*: ⟨ ⊢! (([*r*]! (*p* ⟶! *q*)) ⟷! ([*r*]! *p* ⟶! [*r*]! *q*)) ⟩
| *PK*: ⟨ ⊢! (([*r*]! *K*! *i p*) ⟷! (*r* ⟶! *K*! *i* ([*r*]! *p*))) ⟩

Rules PA1, PA2, PR1 and PR2 are analogous to the rules A1, A2, R1 and R2 of epistemic logic (*ptautology* is implemented in the same style as *tautology*). In addition the system has six axioms – one for each combination of $[\_]_!$ with $\bot_!$, atomic formulas, $\vee_!$, $\wedge_!$, $\longrightarrow_!$ and $K_!$. The axioms for the binary connectives simply distribute $[\_]_!$ over each connective, while the ones for $\bot_!$ and atomic formulas rephrase $[\_]_!$ as an implication. The axiom for $[\_]_!$ and knowledge says that "*i* knows *p* after an announcement *r* if and only if the announcement *r*, whenever truthful, is known by *i* to make *p* true." [2].

4.2 Reducing to Epistemic Logic

We implement the reduction from public announcement logic to epistemic logic operationally, as guided by the reduction axioms. We do so in two steps. The first operation, *reduce' r p*, translates the formula $[r]_!p$ into an equivalent formula in epistemic logic when *p* itself is *static*:

primrec *reduce'* :: ⟨ *'i pfm* ⇒ *'i pfm* ⇒ *'i pfm* ⟩ **where**
⟨ *reduce' r* ⊥! = (*r* ⟶! ⊥!) ⟩
| ⟨ *reduce' r* (*Pro*! *x*) = (*r* ⟶! *Pro*! *x*) ⟩
| ⟨ *reduce' r* (*p* ∨! *q*) = (*reduce' r p* ∨! *reduce' r q*) ⟩
| ⟨ *reduce' r* (*p* ∧! *q*) = (*reduce' r p* ∧! *reduce' r q*) ⟩

```
| ‹reduce' r (p ⟶! q) = (reduce' r p ⟶! reduce' r q)›
| ‹reduce' r (K! i p) = (r ⟶! K! i (reduce' r p))›
| ‹reduce' r ([p]! q) = undefined›
```

The second operation, *reduce p*, reduces the PAL-formula p into epistemic logic by recursion over the syntax:

```
primrec reduce :: ‹'i pfm ⇒ 'i pfm› where
 ‹reduce ⊥! = ⊥!›
| ‹reduce (Pro! x) = Pro! x›
| ‹reduce (p ∨! q) = (reduce p ∨! reduce q)›
| ‹reduce (p ∧! q) = (reduce p ∧! reduce q)›
| ‹reduce (p ⟶! q) = (reduce p ⟶! reduce q)›
| ‹reduce (K! i p) = K! i (reduce p)›
| ‹reduce ([r]! p) = reduce' (reduce r) (reduce p)›
```

We stay within the *pfm* type rather than *fm*, even though we do not use the extra constructors, since our axiomatic system is defined over the *pfm* type.

To prove completeness, we must prove that the reduction preserves the semantics. We do so by first considering the basic *reduce'* operation with a *static* target:

```
lemma reduce'-semantics:
 assumes ‹static q›
 shows ‹((M, w ⊨! [p]! (q))) = (M, w ⊨! reduce' p q)›
 using assms by (induct q arbitrary: w) auto
```

With this lemma we can prove that *reduce* preserves the semantics:

```
lemma reduce-semantics: ‹(M, w ⊨! p) = (M, w ⊨! reduce p)›
```

We refer to the formalization for the proof by structural induction.

4.3 Soundness

We prove the proof system sound similar to how we did for epistemic logic:

```
theorem soundness:
 assumes ‹⊢! p›
 shows ‹M, w ⊨! p›
 using assms by (induct p arbitrary: M w rule: PA.induct) (simp-all add: ptautology)
```

The lemma *ptautology* is analogous to the theorem *tautology* from the formalization of epistemic logic.

4.4 Completeness

We prove the proof system complete. Recall that the *static* formulas are those in which $[\_]_!$ does not occur. The proof system is complete for such formulas:

theorem *static-completeness*:
assumes ‹*static p*› ‹∀ (*M* :: (*'i* :: *countable*, *'i fm set*) *kripke*) *w*. *M*, *w* ⊨! *p*›
shows ‹⊢! *p*›

The reason is that

- ⊢! contains all the axioms of ⊢,
- ⊢ is complete, and
- a static formula is straightforwardly a formula of epistemic logic.

With this theorem in place we can prove completeness for all formulas:

theorem *completeness*:
assumes ‹∀ (*M* :: (*'i* :: *countable*, *'i fm set*) *kripke*) *w*. *M*, *w* ⊨! *p*›
shows ‹⊢! *p*›

We do it by proving that if *p* is true in all models then so is the formula *reduce p* since the reduction is sound. The formula *reduce p* does not contain [_]! and is therefore static. By static completeness, *reduce p* is provable, ⊢! *reduce p*. Additionally we prove from the reduction axioms *PDis*, *PCon*, *PImp*, *PK* and *PFF*, *PPro* that ⊢! *p* ⟷! *reduce p*, and thus that *p* is provable, ⊢! *p*.

5 Related Work

For a good overview of the topic of formalizing logical meta-theory we recommend a recent paper by Blanchette [6].

Several frameworks have been developed for proving logical calculi complete. These frameworks allow the reuse of syntax, semantics and proof ideas to formalize logical systems and their soundness and completeness as well as other results:

- Michaelis and Nipkow formalize a bouquet of different proof systems all based on the same syntax for propositional logic [38, 39]. The framework formalizes sequent calculus, natural deduction, Hilbert systems and resolution.
- The framework by Blanchette, Popescu and Traytel allows proofs of soundness and completeness for proof systems for different logics [7, 8, 9, 10]. This is possible because their framework is parameterized on the specific syntax and semantics. In a related paper's supplementary material, Blanchette and Popescu show that a formalized tableau for many-sorted first-order logic in negation normal form with equality fits in the framework. This supplementary material is unfortunately not up to date with recent Isabelle versions.
- A third development is frameworks for proving completeness of saturation provers. Schlichtkrull et al. [59, 62, 63] formalize the completeness of resolution in a generic way that allows for different provers to be built from the development, which is based on the work by Bachmair and Ganzinger [1]. The development is used to show the soundness and completeness of a particular prover using binary resolution and with a specific strategy for removing redundant clauses, but

other provers would also fit [60, 61]. Tourret and Blanchette reformalize this result [68, 69] based on the more general theory of saturation provers by Waldmann et al. [72].

Outside of the mentioned frameworks, a number of self-contained formalizations of sequent calculi in proof assistants appear in the literature:

- Ridge and Margetson [36, 56, 57] formalized in Isabelle/HOL soundness and completeness for a sequent calculus for formulas in negation normal form and with a term language of only variables.
- Braselmann and Koepke [11, 12] formalized in Mizar soundness and completeness of a sequent calculus.
- Schlöder and Koepke [65] formalized its completeness considering also uncountable languages.
- A more exotic result is the formalization by Ilik [27] in Coq of completeness of a sequent calculus with respect to a Kripke-semantics for classical first-order logic [28].

The following formalizations appear if we broaden the scope to include intuitionistic logic:

- Persson [49] formalized in ALF the soundness of a sequent calculus for intuitionistic first-order logic.
- Herbelin, Kim and Lee [26] formalized in Coq the completeness of a sequent calculus for intuitionistic first-order logic restricted to formulas with implication and universal quantification as the only logical symbols. Their formalization applied a Kripke-style semantics.

If we broaden the scope further to look beyond sequent calculi, we can mention several other formalizations:

- Jensen, Larsen, Schlichtkrull and Villadsen [31, 64] formalized in Isabelle/HOL an axiomatic system for classical logic.
- Raffali [52] formalized in Phox natural deduction for classical logic.
- Persson [49] formalized in ALF natural deduction for intuitionistic logic.
- Peltier [48] formalized in Isabelle/HOL superposition.
- Paulson [45, 46, 47] formalized in Isabelle/HOL Gödel's Incompleteness Theorems, but this does not include a completeness proof.
- Popescu and Traytel present a formalization of Gödel's Incompleteness Theorems [51].
- Jensen, Hindriks and Villadsen [29, 30] also present an approach to formalize in Isabelle/HOL a verification framework for agent programs.

Let us now turn to formalizations of modal logic. These logics contain a single necessity operator $\Box$ rather than one K_i for each agent i in a set of agents:

- Bentzen [3] formalized S5 in Lean.
- Neeley [41] formalized modal systems K, T, S4 and S5 in Lean.

In the context of epistemic logic we found two formalizations in Lean of the S5 system for epistemic logic and PAL. We instead opted to formalize K_n. The S5 system extends K_n in that it has a number of additional axioms, and it is sound and complete when considering Kripke models in which the accessibility relation is an equivalence relation rather than any relation. Additionally there is work on a formalization of intuitionistic epistemic logic in Coq.

- Neeley [41, 42] formalized S5 for epistemic logic and public announcement logic in Lean. Her proof system includes an axiom for the composition of public announcement operators, instead of our axiom for distribution over implication (PImp) and our announcement necessitation rule (PR3).
- Li [34] formalized S5 for epistemic logic and public announcement logic in Lean but only formalized the logical equivalence of the reduction axioms, not the completeness of a proof system that includes them.
- Hagemeier [23] is formalizing intuitonistic epistemic logic in Coq. It is presented in a number of slides, memos and draft memos. We look forward to the finished presentation of the work.
- The Twelf distribution [50] includes a formalization in the LF logical framework [25] of a sequent calculus and natural deduction proof system for classical S5.

The Twelf system [50] is worth mentioning by itself. It provides a uniform metalanguage for specifying logics and proof systems and proving meta-theoretical properties like cut-elimination. However, we are not aware of any formalizations of semantic completeness like we present here.

Other interesting proposals for epistemic logic appear in the literature:

- Kądziołka [33] formalized a solution to a puzzle and introduces a logic tailored to the problem that turns out to be very similar to the possible worlds model of epistemic logic.
- Zuojun, Ågotnes and Zhang [74] presented a variant of epistemic logic that adds the notion of secret knowledge as a first-class citizen. The notion of secrets can be defined in terms of the knowledge operator, but a new modality for secrets is introduced. The authors argue that the main principles can be studied this way, for instance when considering a language with an operator for secrets and without the usual knowledge operator.

Our formalizations rely on deep embedding of formulas. In contrast, using a shallow embedding of the logic means that we write formulas directly in the proof assistant's logic. The advantages of a shallow embedding include not having to formalize semantics, and usually the automation has an easier time proving theorems. The advantage of a deep embedding is that we can obtain formalized soundness and completeness theorems, cf. Sect. 2.

- Benzmüller and Paulson [4] formalized in Isabelle/HOL a shallow encoding of modal logic. Gleißner, Steen and Benzmüller [21, 22] showed effective automation for a wide range of modal logics due to the use of a shallow embedding.

- Reiche and Benzmüller [53] formalized in Isabelle/HOL a shallow embedding of PAL.

Giselle Reis [54] sees formalizing logics in proof assistants as one of several ways to facilitate meta-theory. Concretely, she looks at three methods for facilitating meta-theory: Firstly, she considers using linear logic and subexponential linear logic as a framework for meta-theoretical reasoning. The idea is that certain logics can be expressed in the meta-logic of linear logic and subexponential linear logic. These logics allow some meta-theoretical properties to be proved automatically. Secondly, she considers the use of proof assistants to prove meta-theoretical properties – this is similar to our work here. She notes:

> One of the issues when developing proofs of meta-properties by hand is the sheer complexity and number of cases. By implementing these proofs in proof assistants, the computer will not let us skip cases or overlook details.

We share this experience. Reis experienced that using Coq to formalize logics required her to write specific tactics to do parts of the proofs automatically. In our Isabelle formalization we instead relied on the Isar proof language and the built-in tactics of Isabelle. Reis also explains that working in proof assistants can be combined with the approach of using linear logic as a framework: the idea is that linear logic can be formalized in Coq and then one can use this formalized linear logic to prove properties of other logics. Giselle Reis also notes that formalizations of logics require a significant amount of work:

> The fact that each of these works is a publication (or collection of publications) itself is evidence that formalizing meta-theory is far from trivial work and cannot be done as a matter of fact.

We agree with this perspective and see the building of frameworks in proof assistants and formalizing more logics within them as a way to improve this situation. Additionally, improving the proof assistants themselves will help this agenda. Lastly, Reis considers a solution where the computer aids only in parts of the meta-reasoning, which leaves a part to be done by hand. In particular she considers two systems that can be used for this: GAPT [13] (General Architecture for Proof Theory) is a proof theory framework containing common components of proof theory such as data structures, algorithms, parsers and automated deduction. GAPT interfaces to a number of automated reasoning tools and its focus is on transformation and further processing of proofs. Sequoia [55] is a tool for helping with the meta-theory of sequent calculi and which can import and export LaTeX code. Reis concludes that each method has its strengths and weaknesses and also that much work can be done to make them better and easier to use.

6 Concluding Remarks

For artificial intelligence (AI) in general and for natural language processing (NLP) in particular, the interrelationship between logic and information is pivotal [37]:

> There is a bi-directional relation between logic and information. On the one hand, information underlies the intuitive understanding of standard logical notions such as inference (which may be thought of as the process that turns implicit information into explicit information) and computation. On the other hand, logic provides a formal framework for the study of information itself.

We have considered fundamental axiomatic systems for both epistemic logic (EL) and public announcement logic (PAL). Instead of presenting pen-and-paper proofs of soundness and completeness we have used automated reasoning, the Isabelle proof assistant, as a powerful interactive tool. We share the vision of Rob Nederpelt and Herman Geuvers [40, p. 385]:

> In the future, we expect an enormous increase in the use of proof assistants. Our vision is that formalising a mathematical proof may become as easy as writing mathematics in a mathematical text editor such as LaTeX (Lamport, 1985) and that a mathematical proof will only be accepted for publication when it has been formally checked.

But, in fact, we do not need to choose between pen-and-paper and mechanically checked proofs, as they can successfully coexist.

Acknowledgements We thank Frederik Krogsdal Jacobsen for comments on drafts.

References

1. Bachmair, L., Ganzinger, H., McAllester, D.A., Lynch, C.: Resolution theorem proving. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 19–99. Elsevier and MIT Press (2001)
2. Baltag, A., Renne, B.: Dynamic Epistemic Logic. In: E.N. Zalta (ed.) The Stanford Encyclopedia of Philosophy, Winter 2016 edn. Metaphysics Research Lab, Stanford University (2016)
3. Bentzen, B.: A Henkin-style completeness proof for the modal logic S5 (2019). URL http://arxiv.org/abs/1910.01697. CoRR,
4. Benzmüller, C., Paulson, L.C.: Quantified multimodal logics in simple type theory. Logica Universalis **7**(1), 7–20 (2013). DOI 10.1007/s11787-012-0052-y. URL https://doi.org/10.1007/s11787-012-0052-y
5. Berghofer, S.: First-Order Logic According to Fitting. Archive of Formal Proofs (2007). http://isa-afp.org/entries/FOL-Fitting.html
6. Blanchette, J.C.: Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In: A. Mahboubi, M.O. Myreen (eds.) Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019, pp. 1–13. ACM (2019)
7. Blanchette, J.C., Popescu, A., Traytel, D.: Abstract completeness. Archive of Formal Proofs (2014). https://isa-afp.org/entries/Abstract_Completeness.html, Formal proof development
8. Blanchette, J.C., Popescu, A., Traytel, D.: Unified classical logic completeness - A coinductive pearl. In: S. Demri, D. Kapur, C. Weidenbach (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8562, pp. 46–60. Springer (2014)

9. Blanchette, J.C., Popescu, A., Traytel, D.: Abstract soundness. Archive of Formal Proofs (2017). https://isa-afp.org/entries/Abstract_Soundness.html, Formal proof development
10. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. J. Autom. Reason. **58**(1), 149–179 (2017). DOI 10.1007/s10817-016-9391-3
11. Braselmann, P., Koepke, P.: Gödel's completeness theorem. Formalized Mathematics **13**(1), 49–53 (2005)
12. Braselmann, P., Koepke, P.: A sequent calculus for first-order logic. Formalized Mathematics **13**(1), 33–39 (2005)
13. Ebner, G., Hetzl, S., Reis, G., Riener, M., Wolfsteiner, S., Zivota, S.: System description: GAPT 2.0. In: N. Olivetti, A. Tiwari (eds.) Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings, *Lecture Notes in Computer Science*, vol. 9706, pp. 293–301. Springer (2016). DOI 10.1007/978-3-319-40229-1_20
14. Fagin, R., Halpern, J.Y., Vardi, M.Y., Moses, Y.: Reasoning about Knowledge. MIT Press (1995)
15. Fitting, M.: First-Order Logic and Automated Theorem Proving, Second Edition. Graduate Texts in Computer Science. Springer (1996)
16. From, A.H.: Epistemic logic. Archive of Formal Proofs (2018). https://isa-afp.org/entries/Epistemic_Logic.html, Formal proof development
17. From, A.H.: Public announcement logic. Archive of Formal Proofs (2021). https://isa-afp.org/entries/Public_Announcement_Logic.html, Formal proof development
18. From, A.H., Eschen, A.M., Villadsen, J.: Formalizing axiomatic systems for propositional logic in Isabelle/HOL. In: F. Kamareddine, C. Sacerdoti Coen (eds.) Intelligent Computer Mathematics - 14th International Conference, CICM 2021, Timisoara, Romania, July 26-31, 2021, Proceedings, *Lecture Notes in Artificial Intelligence*, vol. 12833, pp. 32–46. Springer (2021)
19. From, A.H., Jensen, A.B., Villadsen, J.: Formalized soundness and completeness of epistemic logic. In: LAMAS 2021 - 11th Workshop on Logical Aspects of Multi-Agent Systems (2021)
20. From, A.H., Lund, S.T., Villadsen, J.: A case study in computer-assisted meta-reasoning. In: Special Session on Computational Linguistics, Information, Reasoning, and AI 2021 (CompLingInfoReasAI'21), *Lecture Notes in Networks and Systems*, vol. 332, pp. 53–63. Springer (2021). 18th International Conference Distributed Computing and Artificial Intelligence
21. Gleißner, T., Steen, A.: The MET: the art of flexible reasoning with modalities. In: C. Benzmüller, F. Ricca, X. Parent, D. Roman (eds.) Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 11092, pp. 274–284. Springer (2018). DOI 10.1007/978-3-319-99906-7_19. URL https://doi.org/10.1007/978-3-319-99906-7_19
22. Gleißner, T., Steen, A., Benzmüller, C.: Theorem provers for every normal modal logic. In: T. Eiter, D. Sands (eds.) LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017, *EPiC Series in Computing*, vol. 46, pp. 14–30. EasyChair (2017). URL https://easychair.org/publications/paper/6bjv
23. Hagemeier, C.: Formalizing intuitionistic epistemic logic in Coq (2021). URL https://www.ps.uni-saarland.de/~hagemeier/bachelor.php. BSc thesis.
24. Hales, T.C., et al.: A formal proof of the Kepler conjecture. Forum of Mathematics, Pi **5**, 1–29 (2017). DOI 10.1017/fmp.2017.1
25. Harper, R., Honsell, F., Plotkin, G.D.: A framework for defining logics. J. ACM **40**(1), 143–184 (1993). DOI 10.1145/138027.138060
26. Herbelin, H., Kim, S.Y., Lee, G.: Formalizing the meta-theory of first-order predicate logic. Journal of the Korean Mathematical Society **54**(5), 1521–1536 (2017)
27. Ilik, D.: Constructive completeness proofs and delimited control. Ph.D. thesis, École Polytechnique (2010). https://tel.archives-ouvertes.fr/tel-00529021/document
28. Ilik, D., Lee, G., Herbelin, H.: Kripke models for classical logic. Annals of Pure and Applied Logic **161**(11), 1367–1378 (2010)

29. Jensen, A.B.: Towards Verifying GOAL Agents in Isabelle/HOL. In: ICAART 2021 – Proceedings of the 13th International Conference on Agents and Artificial Intelligence – Volume 1, pp. 345–352. SciTePress (2021)
30. Jensen, A.B., Hindriks, K.V., Villadsen, J.: On Using Theorem Proving for Cognitive Agent-Oriented Programming. In: ICAART 2021 – Proceedings of the 13th International Conference on Agents and Artificial Intelligence – Volume 1, pp. 446–453. SciTePress (2021)
31. Jensen, A.B., Larsen, J.B., Schlichtkrull, A., Villadsen, J.: Programming and verifying a declarative first-order prover in Isabelle/HOL. AI Communications **31**(3), 281–299 (2018)
32. Krauss, A.: Defining Recursive Functions in Isabelle/HOL (2021). https://isabelle.in.tum.de/doc/functions.pdf
33. Kądziołka, J.: Solution to the xkcd blue eyes puzzle. Archive of Formal Proofs (2021). https://isa-afp.org/entries/Blue_Eyes.html, Formal proof development
34. Li, J.: Formalization of PAL·S5 in proof assistant. CoRR (2020). URL https://arxiv.org/abs/2012.09388
35. Manna, Z., Pnueli, A.: Formalization of properties of functional programs. J. ACM **17**(3), 555–569 (1970). DOI 10.1145/321592.321606
36. Margetson, J., Ridge, T.: Completeness theorem. Archive of Formal Proofs (2004). http://isa-afp.org/entries/Completeness.html, Formal proof development
37. Martinez, M., Sequoiah-Grayson, S.: Logic and Information. In: E.N. Zalta (ed.) The Stanford Encyclopedia of Philosophy, Spring 2019 edn. Metaphysics Research Lab, Stanford University (2019)
38. Michaelis, J., Nipkow, T.: Formalized proof systems for propositional logic. In: A. Abel, F.N. Forsberg, A. Kaposi (eds.) 23rd International Conference on Types for Proofs and Programs, TYPES 2017, May 29-June 1, 2017, Budapest, Hungary, *LIPIcs*, vol. 104, pp. 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
39. Michaelis, J., Nipkow, T.: Propositional proof systems. Archive of Formal Proofs (2017). http://isa-afp.org/entries/Propositional_Proof_Systems.html, Formal proof development
40. Nederpelt, R., Geuvers, H.: Type Theory and Formal Proof: An Introduction. Cambridge University Press (2014). DOI 10.1017/CBO9781139567725
41. Neeley, P.: A formalization of dynamic epistemic logic. Master's thesis, Carnegie Mellon University (2021). URL https://paulaneeley.com/wp-content/uploads/2021/05/draft1.pdf
42. Neeley, P.: Results in modal and dynamic epistemic logic: A formalization in Lean. Slides Lean Together Workshop (2021). URL https://leanprover-community.github.io/lt2021/slides/paula-LeanTogether2021.pdf
43. Nipkow, T.: Programming and Proving in Isabelle/HOL (2021). https://isabelle.in.tum.de/doc/prog-prove.pdf
44. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, *LNCS*, vol. 2283. Springer (2002)
45. Paulson, L.C.: Gödel's incompleteness theorems. Archive of Formal Proofs (2013). http://isa-afp.org/entries/Incompleteness.html, Formal proof development
46. Paulson, L.C.: A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily finite sets. Rev. Symb. Log. **7**(3), 484–498 (2014). DOI 10.1017/S1755020314000112
47. Paulson, L.C.: A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. J. Autom. Reason. **55**(1), 1–37 (2015)
48. Peltier, N.: A variant of the superposition calculus. Archive of Formal Proofs (2016). http://isa-afp.org/entries/SuperCalc.shtml, Formal proof development
49. Persson, H.: Constructive completeness of intuitionistic predicate logic. Ph.D. thesis, Chalmers University of Technology (1996). http://web.archive.org/web/20001011101511/http://www.cs.chalmers.se/~henrikp/Lic/
50. Pfenning, F., Schürmann, C.: System description: Twelf - A meta-logical framework for deductive systems. In: H. Ganzinger (ed.) Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings, *Lecture Notes in Computer Science*, vol. 1632, pp. 202–206. Springer (1999). DOI 10.1007/3-540-48660-7_14

51. Popescu, A., Traytel, D.: A formally verified abstract account of Gödel's incompleteness theorems. In: P. Fontaine (ed.) Automated Deduction – CADE 27, pp. 442–461. Springer International Publishing, Cham (2019)
52. Raffalli, C.: Krivine's abstract completeness proof for classical predicate logic. https://github.com/craff/phox/blob/master/examples/complete.phx (2005, possibly earlier)
53. Reiche, S., Benzmüller, C.: Public announcement logic in HOL. In: M.A. Martins, I. Sedlár (eds.) Dynamic Logic. New Trends and Applications - Third International Workshop, DaLí 2020, Prague, Czech Republic, October 9-10, 2020, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 12569, pp. 222–238. Springer (2020). DOI 10.1007/978-3-030-65840-3_14. URL https://doi.org/10.1007/978-3-030-65840-3_14
54. Reis, G.: Facilitating meta-theory reasoning (invited paper). In: E. Pimentel, E. Tassi (eds.) Proceedings Sixteenth Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, Pittsburgh, USA, 16th July 2021, *Electronic Proceedings in Theoretical Computer Science*, vol. 337, pp. 1–12. Open Publishing Association (2021). DOI 10.4204/EPTCS.337.1
55. Reis, G., Naeem, Z., Hashim, M.: Sequoia: A playground for logicians - (system description). In: N. Peltier, V. Sofronie-Stokkermans (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 12167, pp. 480–488. Springer (2020). DOI 10.1007/978-3-030-51054-1_32
56. Ridge, T.: A mechanically verified, efficient, sound and complete theorem prover for first order logic. Archive of Formal Proofs (2004). http://isa-afp.org/entries/Verified-Prover.shtml, Formal proof development
57. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings, pp. 294–309 (2005)
58. Ringer, T., Palmskog, K., Sergey, I., Gligoric, M., Tatlock, Z.: QED at large: A survey of engineering of formally verified software. Found. Trends Program. Lang. **5**(2-3), 102–281 (2019). DOI 10.1561/2500000045
59. Schlichtkrull, A., Blanchette, J., Traytel, D., Waldmann, U.: Formalizing Bachmair and Ganzinger's ordered resolution prover. J. Autom. Reason. **64**(7), 1169–1195 (2020)
60. Schlichtkrull, A., Blanchette, J.C., Traytel, D.: A verified functional implementation of Bachmair and Ganzinger's ordered resolution prover. Archive of Formal Proofs (2018). https://isa-afp.org/entries/Functional_Ordered_Resolution_Prover.html, Formal proof development
61. Schlichtkrull, A., Blanchette, J.C., Traytel, D.: A verified prover based on ordered resolution. In: A. Mahboubi, M.O. Myreen (eds.) Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019, pp. 152–165. ACM (2019)
62. Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalization of Bachmair and Ganzinger's ordered resolution prover. Archive of Formal Proofs (2018). https://isa-afp.org/entries/Ordered_Resolution_Prover.html, Formal proof development
63. Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalizing Bachmair and Ganzinger's ordered resolution prover. In: D. Galmiche, S. Schulz, R. Sebastiani (eds.) Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 10900, pp. 89–107. Springer (2018)
64. Schlichtkrull, A., Villadsen, J., From, A.H.: Students' Proof Assistant (SPA). In: P. Quaresma, W. Neuper (eds.) Proceedings 7th International Workshop on Theorem proving components for Educational Software (ThEdu), *EPTCS*, vol. 290, pp. 1–13 (2019)
65. Schlöder, J.J., Koepke, P.: The Gödel completeness theorem for uncountable languages. Formalized Mathematics **20**(3), 199–203 (2012)
66. Smullyan, R.M.: First-Order Logic. Springer-Verlag (1968)
67. Tarski, A.: Logic, Semantics, Metamathematics: Papers from 1923 to 1938. Hackett Publishing (1983)

68. Tourret, S.: A comprehensive framework for saturation theorem proving. Archive of Formal Proofs (2020). https://isa-afp.org/entries/Saturation_Framework.html, Formal proof development
69. Tourret, S., Blanchette, J.: A modular Isabelle framework for verifying saturation provers. In: C. Hritcu, A. Popescu (eds.) CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021, pp. 224–237. ACM (2021)
70. Villadsen, J.: A micro prover for teaching automated reasoning. In: Seventh Workshop on Practical Aspects of Automated Reasoning (PAAR 2020) — Presentation Only / Online Papers, pp. 1–12 (2020). URL http://paar2020.gforge.inria.fr/
71. Villadsen, J.: Tautology checkers in Isabelle and Haskell. In: F. Calimeri, S. Perri, E. Zumpano (eds.) Proceedings of the 35th Edition of the Italian Conference on Computational Logic (CILC 2020), Rende, Italy, 13-15 October 2020, *CEUR Workshop Proceedings*, vol. 2710, pp. 327–341. CEUR-WS.org (2020). URL http://ceur-ws.org/Vol-2710/paper-21.pdf
72. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: N. Peltier, V. Sofronie-Stokkermans (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 12166, pp. 316–334. Springer (2020)
73. Wenzel, M.: The Isabelle/Isar Reference Manual (2021). https://isabelle.in.tum.de/doc/isar-ref.pdf
74. Xiong, Z., Ågotnes, T., Zhang, Y.: The logic of secrets. In: LAMAS 2020 - 10th Workshop on Logical Aspects of Multi-Agent Systems (2020)