A Flexible Approach to Argumentation Framework Analysis using Theorem Proving

David Fuenmayor [1] Alexander Steen [1]

University of Luxembourg, Department of Computer Science
6, avenue de la Fonte
L-4364 Esch-sur-Alzette, Luxembourg

Abstract

Argumentation frameworks constitute the central concept of argumentation theory of Dung. In this paper we present a novel and flexible approach of analyzing argumentation frameworks and their semantics based on an encoding into extensional type theory (classical higher-order logic). This representation enables the use of a wide range of interactive and automated higher-order reasoning tools for assessing argumentation frameworks. This includes the generation of labellings (and extensions), the assessment of meta-theoretic properties, the conduction of interactive empirical experiments, and the flexible analysis of argumentation frameworks with interpreted arguments.

Keywords: Abstract Argumentation, Extensional Type Theory, Automated Reasoning, Meta-logical reasoning, Proof Assistants.

1 Introduction

Argumentation theory is an active field of research in AI. Argumentation frameworks [12] constitute the central concept in abstract argumentation, they have many topical applications in, among others, logic programming, multi-agent systems, non-monotonic reasoning. Since the original formulation of Dung, a lot of research was conducted concerning algorithmic procedures, complexity aspects, as well as various extended and related formalisms (cf. [3] and references therein; cf. also [2] for a comprehensive survey).

In this paper, we propose to investigate argumentation frameworks from the perspective of extensional type theory (ExTT), also commonly simply referred to as *higher-order logic*. To that end, we present an encoding [2] of ar-

[1] E-Mail: {david.fuenmayor, alexander.steen}@uni.lu. Authors are sorted alphabetically by last name. Both authors acknowledge financial support from the Luxembourg National Research Fund (FNR) under grant CORE AuReLeE (C20/IS/14616644).

[2] This is analogous to a *shallow semantical embedding* [4]. We encode the semantics of an *object logic* (in this case: formal argumentation notions) as syntactic abbreviations involving ExTT expressions. *Deep embeddings*, by contrast, explicitly introduce recursive structures representing object-logical formulas together with inductively defined predicates (e.g. *eval*).

gumentation frameworks and their semantics into higher-order logic and make use of available reasoning tools from the (higher-order) automated reasoning community. In particular, we demonstrate how this setup can be used as a framework to (a) flexibly synthesize labellings for argumentation frameworks that satisfy arbitrarily complex properties, (b) conduct (explorative) analyses of meta-logical properties, and (c) formalize rich instantiations of argumentation frameworks in which the arguments can be formulas of some expressive (non-classical) logic.

The experiments presented in this paper were conducted using the proof assistant Isabelle/HOL [17] (cf. §2.2 for some details). The corresponding Isabelle/HOL source files (so-called *theory files*) for this work are freely available at GitHub [14].

The layout of the paper is as follows: In §2 we briefly introduce the concepts of argumentation frameworks and some introductory information on ExTT. §3 presents the encoding of argumentation frameworks and their semantics into higher-order logic. Subsequently, an emphasis is put on the generation of labellings in §4, and an outlook of further application perspectives in given in §5. Finally, we briefly conclude in §6.

2 Preliminaries

The notion of argumentation frameworks and their semantics are introduced. Also, a brief exposition to extensional type theory is given. In the remainder of this paper, the latter formalism will be used for modeling the former.

2.1 Abstract Argumentation

In abstract argumentation theory of Dung [12], arguments are represented as abstract objects and constitute the nodes of a directed graph. The edges of this graph represent (directed) attacks between arguments. This is formalized by the well-known structure of argumentation frameworks. [3]

Definition 2.1 An *argumentation framework* AF is a pair $AF = (A, \rightharpoonup)$, where A is a finite set and $\rightharpoonup \subseteq A \times A$ is a binary relation on A. The elements of A are called *arguments*, and $\rightharpoonup$ is also referred to as the *attack relation*.

An argumentation framework essentially gives an overview of relevant arguments and how they interact (e.g., conflict). Given an argumentation framework AF, one of the main tasks is to determine the subsets of arguments that can be reasonably accepted and those that have to be rejected. This is addressed by so-called argumentation semantics that impose certain restrictions on this selection. There are two standard approaches for argumentation semantics: The more traditional extension-based semantics [12] and the popular labelling-based semantics [1]:

[3] This brief introduction largely follows the survey of Baroni, Caminada and Giacomin [1], to which we refer to for further details on argumentation frameworks and their semantics.

Definition 2.2 An *extension-based semantics* $\mathcal{S}$ associates with each argumentation framework $AF = (A, \rightharpoonup)$ a set of *extensions*, denoted $\mathcal{E}_S(AF)$, where $\mathcal{E}_S(AF) \subseteq 2^A$.

Roughly speaking, an extension is the subset of all arguments that are accepted (under some criterion given by $\mathcal{S}$), while the others are rejected.

Definition 2.3 Let $AF = (A, \rightharpoonup)$ be an argumentation framework. A *labelling* of AF is a function $\mathcal{L}ab : A \Rightarrow \{\texttt{In}, \texttt{Out}, \texttt{Undec}\}$, the set of all labellings of AF is denoted $\mathfrak{L}(AF)$. A *labelling-based semantics* $\mathcal{S}$ then associates with each AF a set of *labellings*, denoted $\mathcal{L}_S(AF)$, where $\mathcal{L}_S(AF) \subseteq \mathfrak{L}(AF)$.

Intuitively, the labels `In` and `Out` represent the status of accepting and rejecting a given argument, respectively. Arguments labelled `Undec` are left undecided, either because one explicitly refrains from accepting resp. rejecting it, or because it cannot be labelled otherwise.

The classical (extension-based) argumentation semantics of Dung are called *conflict-free*, *admissible*, *complete*, *grounded*, *preferred* and *stable* [12]. Furthers include, e.g., *ideal* semantics [13] and *semi-stable* semantics [9]. For each of these semantics there exists an equivalent labelling-based formulation, and each extension can be translated into a labelling and vice versa [10,1].

We omit the formal definitions of the semantics at this point, as they will be subject of the discussions in §3.

2.2 Extensional Type Theory

Extensional type theory (ExTT) is an expressive higher-order logical formalism based on a simply typed λ-calculus which originates from works of Church, Henkin and others [11,15].

The term *higher-order* refers to the ability of ExTT to represent quantifications over predicate and function variables — as opposed to first-order logics, in which quantification is restricted to individuals only. Furthermore ExTT provides λ-notation as an expressive binding mechanism to denote unnamed functions, predicates and sets (by their characteristic functions), and it comes with built-in principles of Boolean and functional extensionality as well as type-restricted comprehension. ExTT constitutes the foundation of most contemporary higher-order automated reasoning systems [6].

Reasoning Systems. Interactive theorem provers (also referred to as *proof assistants*) are systems that allow for the creation and assessment of computer-verified formal proofs, and also facilitate interactive experiments on given formal representations. They are usually based on (extensions of) higher-order logic; one well-established example is Isabelle/HOL [17] that is employed in a wide range of applications, including this paper.

One of the most relevant practical features of Isabelle/HOL is the Sledgehammer system [7] that bridges between the proof assistant and external automated theorem proving (ATP) systems, such as the first-order ATP system E [18] or the higher-order ATP system Leo-III [19]. The idea is to use these automated systems to autonomously solve proof obligations and to import the proofs into the verified context of Isabelle/HOL. The employment of Sledgehammer is of great practical importance and usually a large amount of laborious proof engineering work can be solved by the ATP systems. [4]

3 Encoding of Argumentation Semantics

In this section we present the encoding of argumentation semantics in ExTT, using a syntactical representation close to the one of Isabelle/HOL.

A few technical remarks are in order: In Isabelle/HOL types are either base types, type variables or functional types (the type of functions). In the remainder, o denotes the type of Booleans (i.e., formulas), and $`a$ is a type variable representing an arbitrary type. Function types are denoted $\tau \Rightarrow \nu$, where τ and ν are themselves types. The usual classical connectives are given by $\neg$, $\vee$, $\wedge$, $\longrightarrow$ and $\longleftrightarrow$ for negation, disjunction, conjunction, implication and equivalence, respectively. Universally and existentially quantified formulas are denoted by $\forall X.\, s$ and $\exists X.\, s$, respectively, and anonymous functions are written $\lambda X.\, s$ (where X is an arbitrary identifier and s is a formula).

Further interpreted syntactical notions can be introduced using Isabelle/HOL's meta-logical theory file syntax: A *definition* defines a new symbol that can be regarded (for the purposes of this paper) an abbreviation for terms; we will write $c := s$ to denote the introduction of a new symbol c with definition s, where s is some term, in the following. A *type synonym* is similar to a term definition but rather introduces a new type symbol that abbreviated a (complex) type expression.

Isabelle/HOL formalizes proofs using the general purpose proof language *Isar* [20] which is part of the meta-logical language level of the system. Such formal and internally verified proofs might also be generated by Sledgehammer using external ATP systems. Additionally, counter-models finders such as *Nitpick* [8] may be used to refute given conjectures by providing, if successful, specific counterexamples.

3.1 Basic Notions on Sets and Orderings

The definitions and results discussed below are found in [14, theory `base.thy`].

We start by defining useful type synonyms for the types of sets and relations, which will be represented by characteristic functions (i.e., predicates) on objects of some type $`a$. We thus define $`a$ `Set` and $`a$ `Rel` as type synonyms for the function types $`a \Rightarrow o$ and $`a \Rightarrow `a \Rightarrow o$, respectively. Set equality and the subset relation can be defined as terms of type $`a\ \texttt{Set} \Rightarrow `a\ \texttt{Set} \Rightarrow o$ as follows (all set operations are written as infix operators in the remainder):

$$A \approx B \ := \ \forall x.\ (A\ x) \longleftrightarrow (B\ x) \qquad\qquad A \subseteq B \ := \ \forall x.\ (A\ x) \longrightarrow (B\ x)$$

Analogously, the remaining set-theoretic operations can be defined by anony-

[4] In fact, all of the formalized proofs within Isabelle/HOL presented in the remainder of this paper were generated automatically using Sledgehammer.

mous functions reducing it to the respective underlying logical connectives:

$$A \cap B \;:=\; \lambda w.\ (A\ w) \wedge (B\ w) \qquad\qquad A \cup B \;:=\; \lambda w.\ (A\ w) \vee (B\ w)$$
$$-A \;:=\; \lambda w.\ \neg(A\ w)$$

where $\cap$ and $\cup$ are both terms of type $`a$ `Set` $\Rightarrow$ $`a$ `Set` $\Rightarrow$ $`a$ `Set` and $-$ is of type $`a$ `Set` $\Rightarrow$ $`a$ `Set`.

Because of their importance in various argumentation semantics, we additionally provide generic notions for representing minimal and maximal (resp. least and greatest) sets, with respect to set inclusion: Let Obj be a term of some type τ, and $Prop$ a predicate of type $\tau \Rightarrow o$. We formalize the statement that the set $S(Obj)$ induced by Obj is minimal/maximal/least/greatest among all objects O satisfying property $Prop$ as follows:

$$\begin{aligned}
\mathit{minimal\ Prop\ Obj\ S} \;&:=\; \mathit{Prop\ Obj} \wedge \\
&\quad\ (\forall O.\ \mathit{Prop}\ O \,\wedge\, S(O) \subseteq S(\mathit{Obj}) \longrightarrow S(O) \approx S(\mathit{Obj})) \\
\mathit{maximal\ Prop\ Obj\ S} \;&:=\; \mathit{Prop\ Obj} \wedge \\
&\quad\ (\forall O.\ \mathit{Prop}\ O \,\wedge\, S(\mathit{Obj}) \subseteq S(O) \longrightarrow S(O) \approx S(\mathit{Obj})) \\
\mathit{least\ Prop\ Obj\ S} \;&:=\; \mathit{Prop\ Obj} \wedge \\
&\quad\ (\forall O.\ \mathit{Prop}\ O \,\longrightarrow\, S(\mathit{Obj}) \subseteq S(O)) \\
\mathit{greatest\ Prop\ Obj\ S} \;&:=\; \mathit{Prop\ Obj} \wedge \\
&\quad\ (\forall O.\ \mathit{Prop}\ O \,\longrightarrow\, S(O) \subseteq S(\mathit{Obj}))
\end{aligned}$$

In fact, we formally verified in Isabelle/HOL that, based on these definitions, a least (resp. greatest) set is minimal (resp. maximal) while obtaining countermodels for the converse (using Nitpick). Also, it can be shown that least and greatest elements are unique and that minimal/maximal elements collapse to the least and greatest one when the latter exist. We have also verified some useful results concerning the existence of least/greatest fixed points of monotone functions.

3.2 Extension-based semantics

The definitions and results discussed below are found in [14, theory `extensions.thy`].

In ExTT, an argumentation framework AF is completely characterized by its underlying attack relation $\rightarrow$ of type $`a$ `Rel`, since the set of arguments (i.e., the carrier of $\rightarrow$) is given implicitly as the set of objects of type $`a$. We thus use the type synonym $`a\ \mathcal{AF}$ for argumentation frames as shorthand for $`a$ `Rel`; equaling to type of the underlying attack relation. We will implicitly assume in the following that, in the context of an argumentation framework AF of type $`a\ \mathcal{AF}$, a set of arguments is a term of type $`a$ `Set`.

Given an argumentation framework AF and a set of arguments S, we define the set of attacked and attacking arguments, denoted $[AF|S]^+$ and $[AF|S]^-$, respectively, as follows:

$$[AF|S]^+ := \{b \mid \exists a.\ S\ a \,\wedge\, AF\ a\ b\} \qquad [AF|S]^- := \{b \mid \exists a.\ S\ a \,\wedge\, AF\ b\ a\}.$$

We can now define the fundamental notion of *defense* (aka. *acceptability* in [12]) of arguments:

Definition 3.1 Let A be an argument and S a set of arguments. We say that S *defends* A if each argument B attacking A is itself attacked by at least one argument in S.

This is encoded as a term of type $`a\ \mathcal{AF} \Rightarrow `a\ \texttt{Set} \Rightarrow `a \Rightarrow o$ as follows:

$$defends\ AF\ S\ a\ :=\ \forall b.\, AF\ b\ a \longrightarrow (\exists z.\ S\ z \wedge AF\ z\ b)$$

In fact, Isabelle's simplifier can verify automatically that this definition corresponds to $[AF|\{a\}]^{-} \subseteq [AF|S]^{+}$.

The notion of a *characteristic function* $\mathcal{F}$ of an argumentation framework AF can in fact simply be defined as an alias for the function *defends*, which gives us:

$$\mathcal{F}\ AF\ S\ =\ \lambda a.\, defends\ AF\ S\ a$$

It can easily be verified in Isabelle/HOL that $\mathcal{F}$ (i.e. *defends*) is indeed a monotone function and that it has both a least and greatest fixed point.

The well-known extension-based semantics of Dung [12] have been encoded based on the definitions above (cf. [14, theory `extensions.thy`] for details). We leave them out here, due to space limitations, and focus only on the labelling-based semantics in the following; however, all results in the remainder are equally applicable to the extension-based semantics.

3.3 Labelling-based semantics

The contents of this section are contained in the theories `labellings.thy` and `tests.thy` in the corresponding Isabelle sources [14].

Note that we have encoded sets (i.e., potential argument *extensions*) as functions mapping objects of an arbitrary type $`a$ (i.e., arguments) to the two-element Boolean type o. Generalising on this, we can now define *labellings* as functions into some arbitrary but finite codomain of *labels*. We follow the usual approach given in Def. 2.3 and assume the a set of three labels $\{\texttt{In}, \texttt{Out}, \texttt{Undec}\}$, by defining the Isabelle/HOL datatype:[5]

$$\texttt{Label} := \texttt{In} \mid \texttt{Out} \mid \texttt{Undec},$$

together with $`a$ `Labelling` as type synonym for the type $`a \Rightarrow \texttt{Label}$.

Definition 3.2 Given a labelling $\mathcal{L}ab$ we write $in(\mathcal{L}ab)$, $out(\mathcal{L}ab)$, and $undec(\mathcal{L}ab)$ (read as *in-set*, *out-set*, *undec-set*, respectively) for the sets of arguments labelled by $\mathcal{L}ab$ as `In`, `Out` and `Undec`, respectively.

We encode this definition in Isabelle/HOL as:

$$\begin{aligned} in(\mathcal{L}ab) &:= \lambda x.\, \mathcal{L}ab(x) = \texttt{In} \\ out(\mathcal{L}ab) &:= \lambda x.\, \mathcal{L}ab(x) = \texttt{Out} \\ undec(\mathcal{L}ab) &:= \lambda x.\, \mathcal{L}ab(x) = \texttt{Undec} \end{aligned}$$

[5] A datatype can be, in turn, encoded into plain ExTT.

Now that we have provided means to represent the *as-is* state of an argument wrt. a given labelling, we can additionally represent a target situation in which an argument is said to be adequately or *legally* labelled.

Definition 3.3 Let a be a argument. a is said to be *legally in* if all of its attackers are labelled `Out`. a is said to be *legally out* if it has at least one attacker that is labelled `In`. a is said to be *legally undecided* if it is neither *legally in* nor *legally out.*

In Isabelle/HOL, we encode the above notions by means of predicates of type $`a\ \mathcal{AF} \Rightarrow `a$ `Labelling` $\Rightarrow `a \Rightarrow o$ as follows:

$$\begin{aligned} \mathit{legallyIn}\ AF\ Lab &:= \lambda a. \forall b. (AF\ b\ a) \longrightarrow \mathit{out}\ Lab\ b \\ \mathit{legallyOut}\ AF\ Lab &:= \lambda a. \exists b. (AF\ b\ a) \longrightarrow \mathit{in}\ Lab\ b \\ \mathit{legallyUndec}\ AF\ Lab &:= \lambda a. \neg(\mathit{legallyIn}\ AF\ Lab\ a) \wedge \neg(\mathit{legallyOut}\ AF\ Lab\ a) \end{aligned}$$

Finally, employing the definitions above, the well-known notions of conflict-free and admissible labellings can be defined.

Definition 3.4 [Conflict-free labelling; cf. [1, Def. 16]] A labelling $\mathcal{L}ab$ is termed *conflict-free* if (i) every `In`-labelled argument is not *legally out*; and (ii) every `Out`-labelled argument is *legally out.*

Definition 3.5 [Admissible labelling; cf. [1, Def. 10]] A labelling $\mathcal{L}ab$ is termed *admissible* if (i) every `In`-labelled argument is *legally in*; and (ii) every `Out`-labelled argument is *legally out.*

The two definitions above have been encoded in Isabelle/HOL as predicates of type $`a\ \mathcal{AF} \Rightarrow `a$ `Labelling` $\Rightarrow o$ as follows:

$$\begin{aligned} \mathit{conflictfree}\ AF\ Lab &:= \forall x. (\mathit{in}\ Lab) \longrightarrow \neg \mathit{legallyOut}\ AF\ Lab) \wedge \\ &\qquad (\mathit{out}\ Lab) \longrightarrow \mathit{legallyOut}\ AF\ Lab \\ \mathit{admissible}\ AF\ Lab &:= \forall x. (\mathit{in}\ Lab) \longrightarrow \mathit{legallyIn}\ AF\ Lab) \wedge \\ &\qquad (\mathit{out}\ Lab) \longrightarrow \mathit{legallyOut}\ AF\ Lab \end{aligned}$$

We can, in fact, employ Isabelle to verify automatically that admissible labellings always exist (e.g., consider a labelling where all arguments are `Undec`) and also that admissible labellings are indeed conflict-free.

Moreover, it can be proven automatically that, for admissible labellings, if an argument is *legally undec* then it is labelled `Undec`, but not the other way round (counter-models provided by Nitpick). Interestingly, one can also verify, again by generating counter-models with Nitpick, that for admissible labellings, a *legally in* (resp. *legally out*) argument is not generally labelled `In` (resp. `Out`). This situation changes, however, when we start considering complete labellings.

Definition 3.6 [Complete labelling; cf. [1, Def. 18]] A labelling $\mathcal{L}ab$ is termed *complete* if (i) it is admissible; and (ii) every `Undec`-labelled argument is *legally undec.*

The corresponding Isabelle/HOL encoding is given by:

$$\begin{aligned} &\mathit{complete}\ \mathit{AF}\ \mathit{Lab} := \mathit{admissible}\ \mathit{AF}\ \mathit{Lab} \wedge \\ &\qquad\qquad (\forall x.\ \mathit{undec}\ \mathit{Lab}\ x \longrightarrow \mathit{legallyUndec}\ \mathit{AF}\ \mathit{Lab}\ x) \end{aligned}$$

Using the Sledgehammer tool from within of Isabelle/HOL it can be proven automatically that for complete labellings, *legally in* (resp. *legally out*) arguments are indeed labelled `In` (resp. `Out`). In fact, this alternative definition for a complete labelling has been verified as a theorem:

$$\begin{aligned} \mathit{complete}\ \mathit{AF}\ \mathit{Lab} = \forall x.(&\mathit{in}\ \mathit{Lab}\ x \longleftrightarrow \mathit{legallyIn}\ \mathit{AF}\ \mathit{Lab}\ x) \wedge \\ (&\mathit{out}\ \mathit{Lab}\ x \longleftrightarrow \mathit{legallyOut}\ \mathit{AF}\ \mathit{Lab}\ x) \end{aligned}$$

Also, we can verify automatically that every complete labelling is admissible but not the other way round (since counter-models are found by Nitpick).

In fact, we can prove that for complete labellings, we have that *in/out-sets* completely determine the labelling, i.e., it holds that

$$\big(\mathit{complete}\ \mathit{AF}\ L1 \wedge \mathit{complete}\ \mathit{AF}\ L2\big) \longrightarrow \big(\mathit{in}\ L1 \approx \mathit{in}\ L2 \longrightarrow (L1 = L2)\big)$$

and

$$\big(\mathit{complete}\ \mathit{AF}\ L1 \wedge \mathit{complete}\ \mathit{AF}\ L2\big) \longrightarrow \big(\mathit{out}\ L1 \approx \mathit{out}\ L2 \longrightarrow (L1 = L2)\big).$$

By generating counterexamples with Nitpick we verified that, in contrast, *undec-sets* do not completely determine the (complete) labellings. Another automatically verified result is the following:

$$(\mathit{complete}\ \mathit{AF}\ L1 \wedge \mathit{complete}\ \mathit{AF}\ L2) \longrightarrow (\mathit{in}\ L1 \subseteq \mathit{in}\ L2 \longleftrightarrow \mathit{out}\ L1 \subseteq \mathit{out}\ L2)$$

We now turn to the notions of minimality and maximality for complete labellings, drawing upon the definitions provided in §3.1. We have verified several properties and interrelations for them. As an example, we have shown that given a complete labelling $\mathcal{L}ab$, minimality of in($\mathcal{L}ab$) is equivalent to minimality of out($\mathcal{L}ab$), i.e., it holds that

$$\mathit{minimal}(\mathit{complete}\ \mathit{AF})\ \mathit{Lab}\ \mathit{in} = \mathit{minimal}(\mathit{complete}\ \mathit{AF})\ \mathit{Lab}\ \mathit{out}.$$

With the results above we are now in a position to discuss labellings where *in-sets* are minimal or maximal. They correspond to the so-called *grounded* resp. *preferred* labellings.

Definition 3.7 [Grounded labelling; cf. [1, Def. 20]] A labelling $\mathcal{L}ab$ is termed *grounded* if it is a (in fact: *the*) complete labelling whose *in-set* is minimal (wrt. set inclusion) among all the complete labellings.

Definition 3.8 [Preferred labelling; cf. [1, Def. 22]] A labelling $\mathcal{L}ab$ is termed *preferred* if it is a complete labelling whose *in-set* is maximal (wrt. set inclusion) among all the complete labellings.

The two definitions above are encoded in Isabelle/HOL as follows:

$$\begin{aligned} &\mathit{grounded}\ AF\ Lab := \mathit{minimal}\ (\mathit{complete}\ AF)\ Lab\ \mathit{in} \\ &\mathit{preferred}\ AF\ Lab := \mathit{maximal}\ (\mathit{complete}\ AF)\ Lab\ \mathit{in} \end{aligned}$$

which, recalling the definitions of minimality/maximality in §3.1 unfolds into

$$\begin{aligned} \mathit{grounded}\ AF\ Lab &:= \mathit{complete}\ AF\ Lab \land \\ &\quad (\forall L.\mathit{complete}\ AF\ L \land \mathit{in}(L) \subseteq \mathit{in}(Lab) \longrightarrow \mathit{in}(L) \approx \mathit{in}(Lab)) \\ \mathit{preferred}\ AF\ Lab &:= \mathit{complete}\ AF\ Lab \land \\ &\quad (\forall L.\mathit{complete}\ AF\ L \land \mathit{in}(Lab) \subseteq \mathit{in}(L) \longrightarrow \mathit{in}(L) \approx \mathit{in}(Lab)). \end{aligned}$$

The following well-known result can easily be verified:

$$\mathit{grounded}\ AF\ Lab = \mathit{least}\ (\mathit{complete}\ AF)\ Lab\ \mathit{in}$$

We now turn to complete labellings in which *undec-sets* must satisfy some minimality requirements: stable and semi-stable labellings.

Definition 3.9 [Stable labelling; cf. [1, Def. 24]] A labelling $\mathcal{L}ab$ is termed *stable* if it is a complete labelling whose *undec-set* is empty, i.e., no argument is labelled `Undec`.

The definition above is encoded in Isabelle/HOL as follows:

$$\mathit{stable}\ AF\ Lab := \mathit{complete}\ AF\ Lab \land (\forall x.Lab(x) \neq \texttt{Undec})$$

Definition 3.10 [Semi-stable labelling; cf. [1, Def. 26]] A labelling $\mathcal{L}ab$ is termed *semi-stable* if it is a complete labelling whose *undec-set* is minimal (wrt. set inclusion) among all the complete labellings.

This definition is encoded in Isabelle/HOL as:

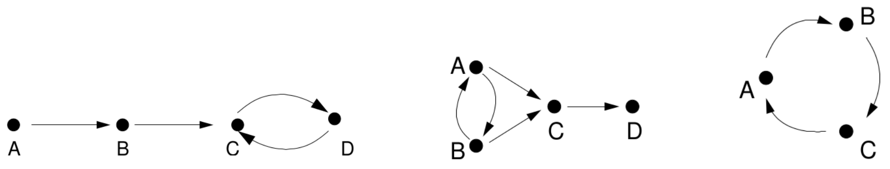
$$\mathit{semistable}\ AF\ Lab := \mathit{minimal}\ (\mathit{complete}\ AF)\ Lab\ \mathit{undec}$$

A complete overview of the encoding of labelling-based semantics in Isabelle is displayed in Fig. A.1 in Appendix B; when disregarding comments and further technical set-up, it only consists of less than 40 lines of code.

We have verified several (meta-)theoretical results for different sorts of complete (preferred, grounded, stable, semi-stable, ideal) labellings in theory `tests.thy`, which we cannot discuss here due to space limitations. In a similar vein, we have encoded and assessed the notions of ideal and stage labellings, and some notions for *skeptical* and *credulous* argument justification (cf. [1, §4]).

4 Flexible Generation of Labellings

The encoding for argumentation semantics presented above captures the structure and the logical behavior of argumentation frameworks within extensional type theory. Building on top of that, we can make use of automation tools from within Isabelle/HOL for generating labellings for concrete argumentation



(a) A simple AF (b) Floating acceptance (c) A odd cycle

Fig. 1. Examples of argumentation frameworks. Graphics by Baroni, Caminada and Giacomin [1, Fig. 4-6].

frameworks. Figure 1 displays a few examples of argumentation frameworks taken from [1] that serve as use cases to illustrate the generation of labellings.

As an example, consider the argumentation framework from Fig. 1a: As a first step we define a new datatype, say `Arg`, for its arguments – consisting only of the distinct terms `A`, `B`, `C` and `D`. Next, we encode the attack relation `att` as binary predicate such that `att` X Y if and only if X attacks Y as displayed in Fig. 1a. The original source of this setup in Isabelle/HOL are displayed in Figure B.1 in Appendix B. We can now use the higher-order model finder Nitpick to ask for a, say, stable labelling. Indeed, Nitpick produces the following labelling (see the original output in Fig. B.2 in Appendix B):

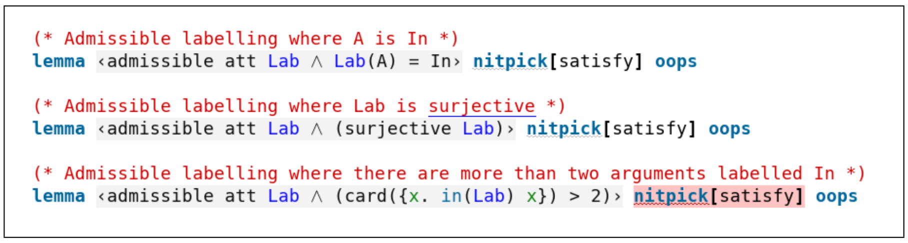
$$Lab = x \mapsto \begin{cases} \texttt{In} & \text{if } x = A \\ \texttt{Out} & \text{if } x = B \\ \texttt{Out} & \text{if } x = C \\ \texttt{In} & \text{if } x = D \end{cases}$$

which represents the labelling Lab such that $in(Lab) = \{A, D\}$, $out(Lab) = \{B, C\}$ and $undec(Lab) = \emptyset$. Even more, we can employ Nitpick to generate all such labellings by just inspecting the value given to the free variable $LabSet$ in the formula below (cf. also Fig. B.3).

$$\forall Lab.\ LabSet\ Lab \longleftrightarrow stable\ att\ Lab$$

The reported labellings are in fact exactly those described in [1]. The same holds for the remaining argumentation semantics and examples from Fig. 1.

In addition to the above – quite standard – applications, we can now make use of the expressive surrounding logical framework to ask for *specific* labellings, e.g., satisfying an arbitrary (higher-order) predicate P. Consider the following example from Isabelle/HOL, relating to the example from Fig. 1b:



```
(* Admissible labelling where A is In *)
lemma ‹admissible att Lab ∧ Lab(A) = In› nitpick[satisfy] oops

(* Admissible labelling where Lab is surjective *)
lemma ‹admissible att Lab ∧ (surjective Lab)› nitpick[satisfy] oops

(* Admissible labelling where there are more than two arguments labelled In *)
lemma ‹admissible att Lab ∧ (card({x. in(Lab) x}) > 2)› nitpick[satisfy] oops
```

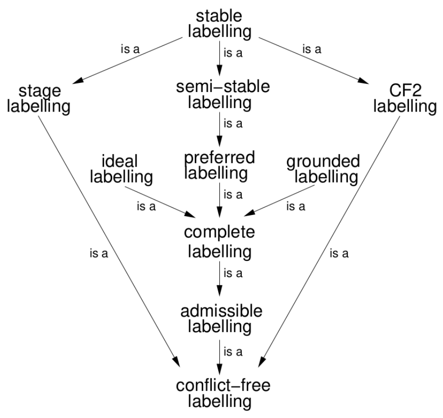


Fig. 2. An overview of the inclusion relations among the well-known labelling semantics. Graphics by Baroni, Caminada and Giacomin [1, Fig. 12].

In the three lemmas, we ask Nitpick to generate admissible labellings where, additionally, (1) argument A is labelled `In`, (2) *Lab* is a surjective function, and (3) there are more than two arguments labelled `In`, respectively. In the first two cases, suitable labellings are provided, in the third case no such labelling can be found (visualized by the red background color indicating an error). Indeed, no such labelling exists.

Similarly, we can prove in Isabelle/HOL that for Fig. 1c no admissible labelling other than the trivial one exists. This is expressed by the formula

$$\textit{admissible att Lab} \longrightarrow \forall x.\, \textit{Lab}(x) = \texttt{Undec}$$

which is proven automatically by Sledgehammer within a few seconds.

5 Application Perspectives

Verification of (Meta-)Logical Properties. In §3 we have mentioned, in passing, some of the results obtained through the use of Isabelle's automated tools (proven theorems and generated counter-models). In fact, the expressivity of HOL allowed us to formalize definitions and theorems involving general high-level mathematical notions such us minimality/maximality, least/greatest, fixed-points, etc., as well as domain-specific notions (e.g. *legally in/out*). These notions can, in turn, be used as building blocks to generate and assess further meta-logical properties (e.g., that for complete labellings in-sets completely determine the labelling). As an illustration, many of the inclusion relationships among the different labelling-based semantics, cf. Fig. 2, have been automatically verified in Isabelle/HOL using Sledgehammer [14, theory `tests.thy`].

Empirical Theory Exploration. In line with the above, a further contribution of the presented work consists in providing computer-assisted support for the (meta-)theoretical analysis of argumentation semantics. Next to verifying well-known results from the literature, (higher-order) theorem proving technology provides expressive means to formalize (meta-)theoretical conjectures and put them to the test by employing the reasoning tools of, e.g., Isabelle/HOL. This way we can explore the conceptual space of a theory in an empirical fashion, while interactively receiving feedback from the software about the adequacy of the formal model (e.g., by generating counterexamples).

Rich Instantiations of Argumentation Frameworks. There exist extended notions of abstract argumentation frameworks in the sense that the arguments are regarded as interpreted structures (as opposed to abstract objects), cf. [3] and the references therein. Using the here presented approach, we can easily interpret the abstract type 'a as complex structures within ExTT and, in turn, instantiate the argumentation framework with these objects. This includes, but is not limited to, taking arguments to be (sets or theories of) modal logic formulas, paraconsistent formulas or deontic logic formulas. For each of these examples there already exist embeddings into higher-order logic and, hence, a combination of these concepts is mostly engineering work.

6 Conclusion

In this paper, we modeled argumentation frameworks as typed binary relations in extensional type theory. Using this modeling, we provided an encoding of the well-known argumentation semantics, both in the extension-based and the labelling-based variant. The presented encoding was implemented using the Isabelle/HOL proof assistant, in which also selected experiments have been illustrated, including the generation of labellings and the verification of meta-logical properties.

The proposed approach is by no means meant (nor able) to challenge well-established and highly efficient procedures for calculating labellings (or extensions) of argumentation frameworks. It rather contributes a novel method of exploring the generation and analysis of argumentation semantics in the context of an expressive logical setting which cannot be addressed by existing approaches.

We argue that, mostly, the encoding re-formulates the basic definitions of argumentation semantics (without any major conceptual modifications) into the formal system of ExTT; and hence a formal proof would be mostly technical. Additionally, the logical and meta-logical experiments yielded the expected results in all cases. Still, a formal proof of faithfulness of the encoding would give a proper grounding of this work. Such a proof is further work.

The proposed approaches are, from a methodological point of view, in line with recent work in the computer-assisted assessment of normative theories using the LogiKEy [5] framework. Combining the here presented approaches and the formalized LogiKEy theories (also conducted in Isabelle/HOL), argumentation networks could be flexibly employed in the definition of ethico-legal governors. Another interesting path would be to incorporate explanation semantics [16] into the presented framework.

We encourage the interested reader to carry out further experiments, and also to further expand and improve on this work.

References

[1] Baroni, P., M. Caminada and M. Giacomin, *An introduction to argumentation semantics*, Knowl. Eng. Rev. **26** (2011), pp. 365–410.

[2] Baroni, P., D. M. Gabbay, M. Giacomin and L. van der Torre, "Handbook of formal argumentation," College Publications, 2018.

[3] Baroni, P., F. Toni and B. Verheij, *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games: 25 years later*, Argument Comput. **11** (2020), pp. 1–14.

[4] Benzmüller, C., *Universal (meta-)logical reasoning: Recent successes*, Science of Computer Programming **172** (2019), pp. 48–62.

[5] Benzmüller, C., X. Parent and L. W. N. van der Torre, *Designing normative theories for ethical and legal reasoning:* LogiKEy *framework, methodology, and tool support*, Artif. Intell. **287** (2020), p. 103348.

[6] Benzmüller, C. and P. Andrews, *Church's Type Theory*, in: E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*, Metaphysics Research Lab, Stanford University, 2019, summer 2019 edition .

[7] Blanchette, J. C., S. Böhme and L. C. Paulson, *Extending sledgehammer with SMT solvers*, J. Autom. Reason. **51** (2013), pp. 109–128.

[8] Blanchette, J. C. and T. Nipkow, *Nitpick: A counterexample generator for higher-order logic based on a relational model finder*, in: M. Kaufmann and L. C. Paulson, editors, *ITP 2010*, LNCS **6172** (2010), pp. 131–146.

[9] Caminada, M. W. A., W. A. Carnielli and P. E. Dunne, *Semi-stable semantics*, J. Log. Comput. **22** (2012), pp. 1207–1254.

[10] Caminada, M. W. A. and D. M. Gabbay, *A logical account of formal argumentation*, Stud Logica **93** (2009), pp. 109–145.

[11] Church, A., *A formulation of the simple theory of types*, J. Symb. Log. **5** (1940), pp. 56–68.

[12] Dung, P. M., *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games*, Artif. Intell. **77** (1995), pp. 321–358.

[13] Dung, P. M., P. Mancarella and F. Toni, *Computing ideal sceptical argumentation*, Artif. Intell. **171** (2007), pp. 642–674.

[14] Fuenmayor, D. and A. Steen, *Isabelle/HOL sources associated with this paper*, Online available at Github: `https://github.com/aureleeNet/formalizations` (2021).

[15] Henkin, L., *Completeness in the theory of types*, J. Symb. Log. **15** (1950), pp. 81–91.

[16] Liao, B. and L. van der Torre, *Explanation semantics for abstract argumentation*, in: H. Prakken, S. Bistarelli, F. Santini and C. Taticchi, editors, *Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020*, Frontiers in Artificial Intelligence and Applications **326** (2020), pp. 271–282.

[17] Nipkow, T., L. C. Paulson and M. Wenzel, "Isabelle/HOL - A Proof Assistant for Higher-Order Logic," Lecture Notes in Computer Science **2283**, Springer, 2002.

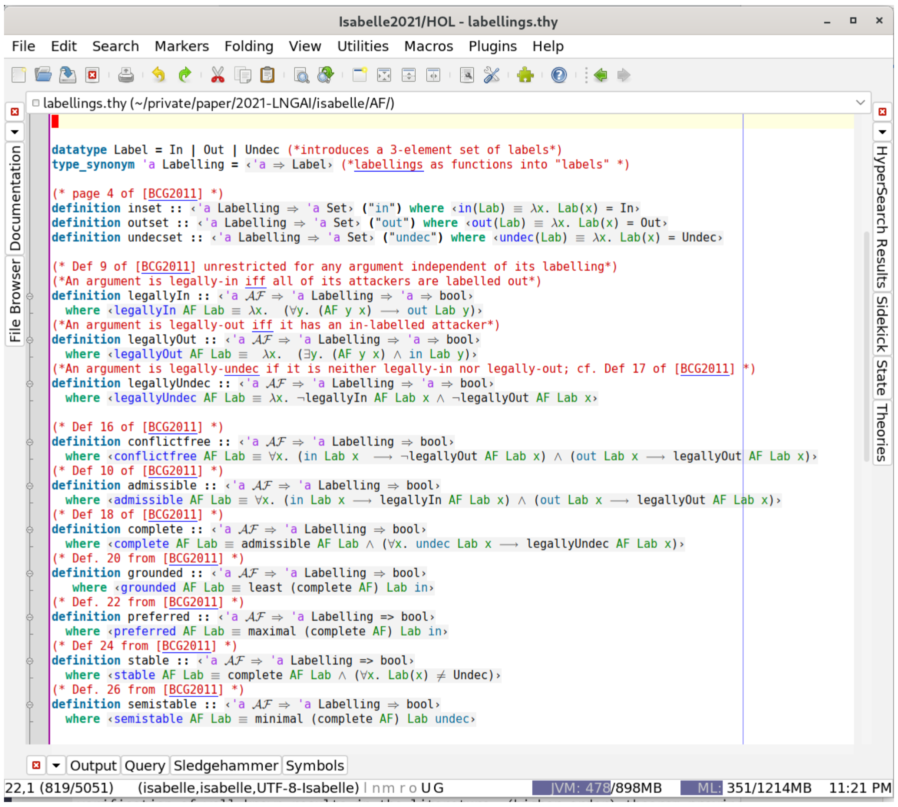
[18] Schulz, S., *E – a brainiac theorem prover*, AI Commun. **15** (2002), pp. 111–126.

[19] Steen, A. and C. Benzmüller, *Extensional higher-order paramodulation in Leo-III*, Journal of Automated Reasoning (2021).

[20] Wenzel, M., *Isabelle/Isar—a generic framework for human-readable proof documents*, From Insight to Proof-Festschrift in Honour of Andrzej Trybulec **10** (2007), pp. 277–298.

Appendix

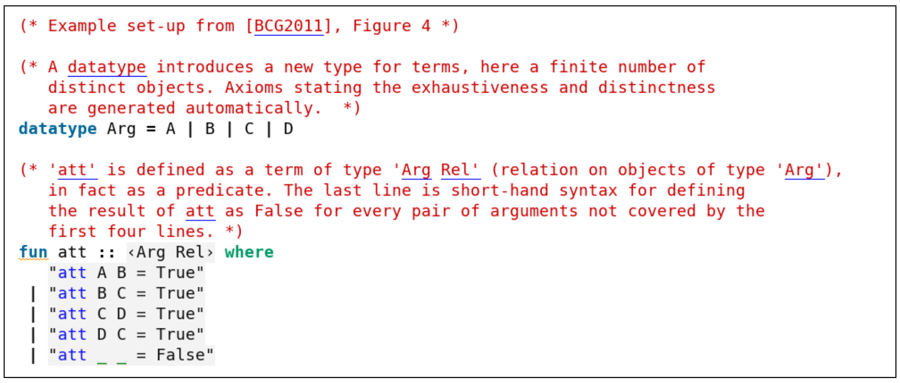
A Isabelle Formalization of Argumentation Frameworks


Isabelle2021/HOL - labellings.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
labellings.thy (~/private/paper/2021-LNGAI/isabelle/AF/)
File Browser
Documentation
HyperSearch Results
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
22,1 (819/5051) (isabelle,isabelle,UTF-8-Isabelle) I nm r o U G JVM: 478/898MB ML: 351/1214MB 11:21 PM

```
datatype Label = In | Out | Undec (*introduces a 3-element set of labels*)
type_synonym 'a Labelling = ‹'a ⇒ Label› (*labellings as functions into "labels" *)

(* page 4 of [BCG2011] *)
definition inset :: ‹'a Labelling ⇒ 'a Set› ("in") where ‹in(Lab) ≡ λx. Lab(x) = In›
definition outset :: ‹'a Labelling ⇒ 'a Set› ("out") where ‹out(Lab) ≡ λx. Lab(x) = Out›
definition undecset :: ‹'a Labelling ⇒ 'a Set› ("undec") where ‹undec(Lab) ≡ λx. Lab(x) = Undec›

(* Def 9 of [BCG2011] unrestricted for any argument independent of its labelling*)
(*An argument is legally-in iff all of its attackers are labelled out*)
definition legallyIn :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ 'a ⇒ bool›
  where ‹legallyIn AF Lab ≡ λx.  (∀y. (AF y x) ⟶ out Lab y)›
(*An argument is legally-out iff it has an in-labelled attacker*)
definition legallyOut :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ 'a ⇒ bool›
  where ‹legallyOut AF Lab ≡  λx.  (∃y. (AF y x) ∧ in Lab y)›
(*An argument is legally-undec if it is neither legally-in nor legally-out; cf. Def 17 of [BCG2011] *)
definition legallyUndec :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ 'a ⇒ bool›
  where ‹legallyUndec AF Lab ≡ λx. ¬legallyIn AF Lab x ∧ ¬legallyOut AF Lab x›

(* Def 16 of [BCG2011] *)
definition conflictfree :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ bool›
  where ‹conflictfree AF Lab ≡ ∀x. (in Lab x  ⟶ ¬legallyOut AF Lab x) ∧ (out Lab x ⟶ legallyOut AF Lab x)›
(* Def 10 of [BCG2011] *)
definition admissible :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ bool›
  where ‹admissible AF Lab ≡ ∀x. (in Lab x ⟶ legallyIn AF Lab x) ∧ (out Lab x ⟶ legallyOut AF Lab x)›
(* Def 18 of [BCG2011] *)
definition complete :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ bool›
  where ‹complete AF Lab ≡ admissible AF Lab ∧ (∀x. undec Lab x ⟶ legallyUndec AF Lab x)›
(* Def. 20 from [BCG2011] *)
definition grounded :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ bool›
   where ‹grounded AF Lab ≡ least (complete AF) Lab in›
(* Def. 22 from [BCG2011] *)
definition preferred :: ‹'a 𝒜ℱ ⇒ 'a Labelling => bool›
  where ‹preferred AF Lab ≡ maximal (complete AF) Lab in›
(* Def 24 from [BCG2011] *)
definition stable :: ‹'a 𝒜ℱ ⇒ 'a Labelling => bool›
  where ‹stable AF Lab ≡ complete AF Lab ∧ (∀x. Lab(x) ≠ Undec)›
(* Def. 26 from [BCG2011] *)
definition semistable :: ‹'a 𝒜ℱ ⇒ 'a Labelling ⇒ bool›
  where ‹semistable AF Lab ≡ minimal (complete AF) Lab undec›
```

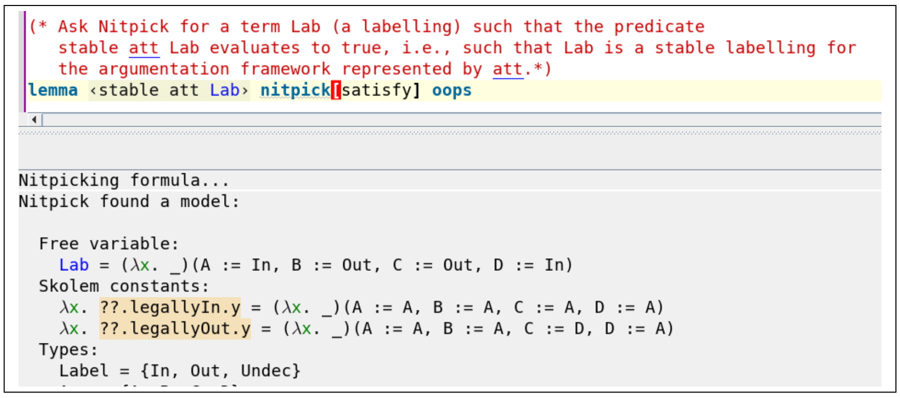
Fig. A.1. Overview of the Isabelle/HOL implementation of labelling-based argumentation semantics. The whole implementation consists of less than 40 lines of code.

B Isabelle Setup for Extensions/Labellings Generation



```
(* Example set-up from [BCG2011], Figure 4 *)

(* A datatype introduces a new type for terms, here a finite number of
   distinct objects. Axioms stating the exhaustiveness and distinctness
   are generated automatically.  *)
datatype Arg = A | B | C | D

(* 'att' is defined as a term of type 'Arg Rel' (relation on objects of type 'Arg'),
   in fact as a predicate. The last line is short-hand syntax for defining
   the result of att as False for every pair of arguments not covered by the
   first four lines. *)
fun att :: ‹Arg Rel› where
   "att A B = True"
 | "att B C = True"
 | "att C D = True"
 | "att D C = True"
 | "att _ _ = False"
```

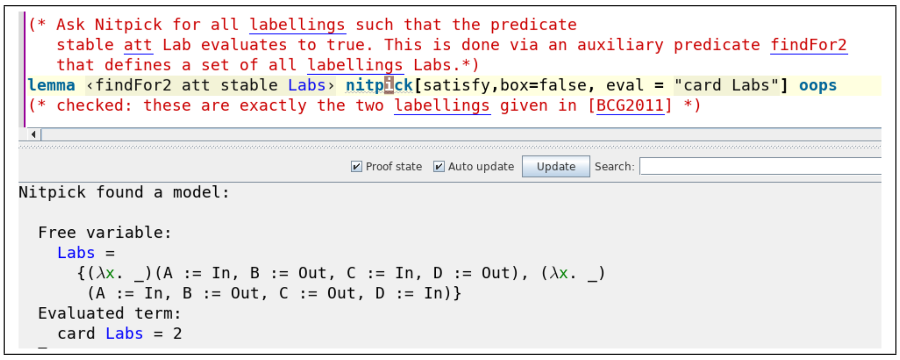
Fig. B.1. Encoding of the argumentation framework from Fig. 1a.



```
(* Ask Nitpick for a term Lab (a labelling) such that the predicate
   stable att Lab evaluates to true, i.e., such that Lab is a stable labelling for
   the argumentation framework represented by att.*)
lemma ‹stable att Lab› nitpick[satisfy] oops
```

```
Nitpicking formula...
Nitpick found a model:

  Free variable:
    Lab = (λx. _)(A := In, B := Out, C := Out, D := In)
  Skolem constants:
    λx. ??.legallyIn.y = (λx. _)(A := A, B := A, C := A, D := A)
    λx. ??.legallyOut.y = (λx. _)(A := A, B := A, C := D, D := A)
  Types:
    Label = {In, Out, Undec}
```

Fig. B.2. Nitpick output for a stable labelling of argumentation framework in Fig. 1a.



```
(* Ask Nitpick for all labellings such that the predicate
   stable att Lab evaluates to true. This is done via an auxiliary predicate findFor2
   that defines a set of all labellings Labs.*)
lemma ‹findFor2 att stable Labs› nitpick[satisfy,box=false, eval = "card Labs"] oops
(* checked: these are exactly the two labellings given in [BCG2011] *)
```

```
Nitpick found a model:

  Free variable:
    Labs =
      {(λx. _)(A := In, B := Out, C := In, D := Out), (λx. _)
        (A := In, B := Out, C := Out, D := In)}
  Evaluated term:
    card Labs = 2
```

Fig. B.3. Nitpick output enumerating all stable labellings of the argumentation framework from Fig. 1a. For this we employed (an optimized variant of) the utility function: $\mathit{findFor}\ AF\ Prop\ S\ :=\ \forall Lab.\ S(Lab) \longleftrightarrow Prop(AF)\ Lab.$