Abstract Argumentation in Higher-Order Logic

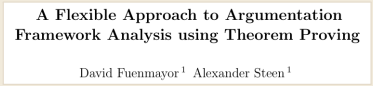
The embedding of Fuenmayor and Steen: a guided tour of the encoding, its meta-theory and its use, on the Isabelle session `AA_Fixed`

Christoph Benzmüller and Luca Pasetto

LogiKEy course, November 2026

Two papers, one library

LNGAI 2021: the first version[1]


A Flexible Approach to Argumentation Framework Analysis using Theorem Proving

David Fuenmayor [1] Alexander Steen [1]

- A framework **is** its attack relation; the arguments are all inhabitants of a type
- Labelling semantics in the foreground
- Labellings generated with the model finder Nitpick
- Perspectives: meta-theory, exploration, instantiated arguments

J. Logic Comput. 2024: the extended version[2]

A. Steen, D. Fuenmayor. *A formalization of abstract argumentation in higher-order logic*

- A framework is a pair $(\mathcal{A}, att)$ with an **explicit universe** $\mathcal{A}$
- Nine semantics, each as extensions and as labellings
- Adequacy: the results of Dung and of Baroni, Caminada and Giacomin, verified
- Theory exploration; the correspondence between extensions and labellings

All Isabelle text on these slides is taken from the session `AA_Fixed`: its main theories use the explicit universe, its folder `simplified/` the whole type.

[1] Fuenmayor and Steen, *A Flexible Approach to Argumentation Framework Analysis using Theorem Proving*, LNGAI 2021.

[2] Steen and Fuenmayor, *A formalization of abstract argumentation in higher-order logic*, Journal of Logic and Computation 34(2) 2024.

Outline

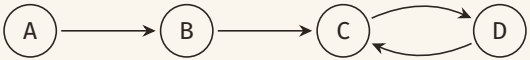
1. Abstract argumentation in ten minutes
2. Why higher-order logic, and which tools
3. The encoding: building blocks
4. The semantics as HOL predicates
5. Second pass: an explicit universe of arguments
6. Reasoning *about* argumentation: the meta-theory
7. Reasoning *with* argumentation: generating extensions and labellings
8. Positioning, outlook, summary

01

Abstract argumentation in ten minutes

Argumentation frameworks

An **argumentation framework** is a pair $(\mathcal{A}, \rightarrow)$: a set of arguments $\mathcal{A}$ and an attack relation $\rightarrow \subseteq \mathcal{A} \times \mathcal{A}$.[3]



The running example of this talk: A attacks B, B attacks C, C and D attack each other. It is Fig. 4 of the survey by Baroni, Caminada and Giacomin,[4] "BCG" from here on.

- Arguments are **abstract**: no internal structure, only the attacks matter. What they stand for depends on the application: defeasible proofs, reasons, legal claims
- The question: which sets of arguments can be **accepted together**?
- An **argumentation semantics** answers it. There are many, and they come in two styles: extensions and labellings

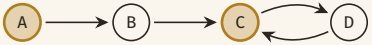
[3] Dung, *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games*, Artificial Intelligence 1995.

[4] Baroni et al., *An introduction to argumentation semantics*, Knowledge Engineering Review 2011.

Extension semantics I: conflict-free, admissible, complete

A set of arguments S ...

- **defends** a: every attacker of a is attacked by some member of S
- is **conflict-free**: no member attacks a member
- is **admissible**: conflict-free, and it defends each of its members
- is **complete**: admissible, and it contains every argument it defends

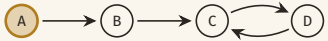


- $\{A, C\}$ is complete: A defends C against B, C defends itself against D

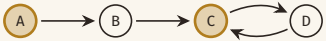
Semantics	Sets in the running example
conflict-free (8)	$\emptyset$, $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{A, C\}$, $\{A, D\}$, $\{B, D\}$
admissible (5)	$\emptyset$, $\{A\}$, $\{D\}$, $\{A, C\}$, $\{A, D\}$
complete (3)	$\{A\}$, $\{A, C\}$, $\{A, D\}$

- $\{B, D\}$ is conflict-free but not admissible: nothing defends B against A
- $\{C\}$ is not admissible either: C does not answer B
- $\emptyset$ and $\{D\}$ are admissible but not complete: A is unattacked, so every set defends it

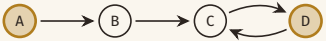
Extension semantics II: grounded, preferred, stable



Grounded: $\{A\}$



Preferred and **stable**: $\{A, C\}$



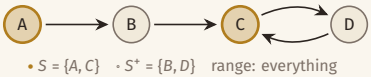
Preferred and **stable**: $\{A, D\}$

- **Grounded** extension: the $\subseteq$-least complete extension. What every complete extension accepts; equivalently, the least fixed point of "the arguments defended by S"
- **Preferred** extension: a $\subseteq$-maximal complete extension. As much as can be defended at once
- **Stable** extension: a conflict-free set that attacks every argument outside it

- The grounded extension always exists and is unique; preferred extensions always exist, possibly several
- Stable extensions need not exist: a three-cycle of attacks has none

Beyond Dung: semi-stable, stage, ideal

- The **range** of S is $S \cup S^+$: the set together with everything it attacks. A stable extension has the whole framework as its range
- **Semi-stable**: a complete extension whose range is $\subseteq$-maximal among the complete extensions
- **Stage**: a conflict-free set whose range is $\subseteq$-maximal among the conflict-free sets
- **Ideal**: the greatest admissible set that is contained in every preferred extension[5]
- Semi-stable and stage are "as stable as possible": they coincide with the stable extensions whenever those exist



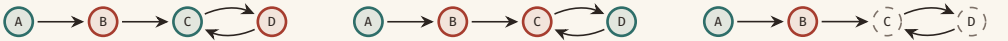
Semantics	Running example
semi-stable	$\{A, C\}$, $\{A, D\}$
stage	$\{A, C\}$, $\{A, D\}$
ideal	$\{A\}$

[5] Verheij, *Two approaches to dialectical argumentation*, NAIC 1996; Caminada, *Semi-stable semantics*, COMMA 2006; Dung et al., *Computing ideal sceptical argumentation*, Artificial Intelligence 2007.

Labellings: In, Out, Undec

A **labelling** is a total function *Lab* from $\mathcal{A}$ to {**In**, **Out**, **Undec**}: accepted, rejected, undecided.[6]

- An argument is **legally in** if all its attackers are **Out**; **legally out** if some attacker is **In**; **legally undecided** if neither
- A **complete labelling** gives every argument exactly the label it is legally entitled to



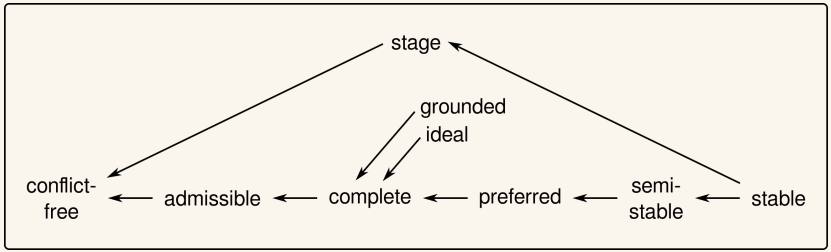
in-set {*A*, *C*) in-set {*A*, *D*} in-set {*A*}: the grounded labelling

The three complete labellings of the running example. • In • Out • Undec

- Every extension semantics has a labelling version; the in-sets of the complete labellings are exactly the complete extensions
- A labelling says more than an extension: it separates rejected from undecided arguments

[6] Caminada, *On the issue of reinstatement in argumentation*, JELIA 2006; Caminada and Gabbay, *A logical account of formal argumentation*, Studia Logica 2009.

The landscape of semantics



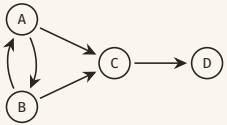
- An arrow reads **“is a”**: every stable extension is semi-stable, every preferred extension is complete, and so on. Arrows that follow by transitivity are omitted
- Each name stands for both the extension and the labelling version. No arrow can be reversed
- This picture returns in part 6 as a **list of proof obligations**: one lemma per arrow, one countermodel per missing arrow

Figure: Steen & Fuenmayor, J. Logic and Computation 34(2), 2024, after BCG, Fig. 12.

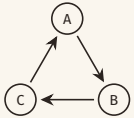
Three benchmark graphs



Simple AF (BCG Fig. 4)



Floating acceptance (BCG Fig. 5)



Odd cycle (BCG Fig. 6)

Extensions	Simple AF	Floating acceptance	Odd cycle
complete	$\{A\}, \{A, C\}, \{A, D\}$	$\emptyset, \{A, D\}, \{B, D\}$	$\emptyset$
grounded, ideal	$\{A\}$	$\emptyset$	$\emptyset$
preferred, semi-stable	$\{A, C\}, \{A, D\}$	$\{A, D\}, \{B, D\}$	$\emptyset$
stable	$\{A, C\}, \{A, D\}$	$\{A, D\}, \{B, D\}$	none

- Floating acceptance: D is in every preferred extension, yet the grounded extension is empty
- $\emptyset$ is one extension with no members; “none” means that there is no extension at all
- The three graphs come back in part 7, where the tools must reproduce this table

02

Why higher-order logic, and which tools

Why put argumentation into a proof assistant?

Reasoning *with* frameworks: the object level

Given this graph, what are its preferred extensions? Is A accepted in all of them?

What solvers do, and do well: SAT, QBF and answer-set programming back ends.

Reasoning *about* frameworks: the meta level

Does every framework have a preferred extension? Is every stable extension preferred? Is this new conjecture true?

Traditionally done with pen and paper.

The claim of the two papers: one logic and one tool chain for both kinds of reasoning

- **Uniform**: definitions, theorems about them, and concrete graphs live in the same language
- **Flexible**: ask for extensions that satisfy any further property you can write down
- **Open**: the arguments can later be instantiated with structured objects, such as formulas
- **Checked**: every proved statement is verified by the kernel of a proof assistant

Logical encodings of argumentation so far

Target formalism	Representative work	Reasoning about the semantics
Propositional logic, SAT	Dunne & Bench-Capon 2002; Besnard & Doutre 2004; Cegartix, LabSATSolver, jArgSemSAT	no: extensions are computed, not studied
Logic programs	Toni & Sergot 2011	no
Quantified Boolean formulas	Egly & Woltran 2006; Arieli & Caminada 2013	some: existence, selected inclusions and equivalences
First-order fragments	YALLA (Dupin de Saint-Cyr et al. 2016); Cayrol & Lagasquie-Schiex 2020	restricted
Modal and second-order logic	surveyed by Gabbay 2013	partly
Higher-order logic	**Fuenmayor & Steen 2021**	**yes, in the same language**

- "Preferred = $\subseteq$-maximal complete" quantifies over **sets of arguments**: in HOL, an ordinary quantifier
- The less expressive encodings stay decidable and scale. The HOL encoding **complements** them

HOL in a nutshell: types and terms

Types

- Base types: o for truth values (`bool` in Isabelle), ι for individuals
- Function types: $\tau \Rightarrow v$
- Type variables: `'a`, for "any type"

Terms

$$s, t \ ::= \ c_\tau \ | \ X_\tau \ | \ (\lambda X_\tau . s_v)_{\tau \Rightarrow v} \ | \ (s_{\tau \Rightarrow v} \, t_\tau)_v$$

constants, variables, abstraction, application. Every term has exactly one type.

Functions take one argument at a time

- `f a b` means `(f a) b`: **currying**
- `f a` alone is a function again: **partial application**
- Functions can be arguments of functions: `g (f a)`

The example to keep in mind

`att :: 'a ⇒ 'a ⇒ bool`

`att a b` is the statement "a attacks b".

`att a` is the set of arguments that a attacks.

- This is Church's simple theory of types (1940), with Henkin's general semantics (1950)
- Also called extensional type theory. "Higher-order": quantifiers may range over functions and predicates

Formulas are terms, sets are predicates

- A **formula** is a term of type `bool`
- The connectives are constants: `∧ :: bool ⇒ bool ⇒ bool`
- So is the quantifier: `∀x. s` abbreviates `∀ (λx. s)`, for a variable `x` of **any** type
- A **set** over `'a` is its characteristic function, a term of type `'a ⇒ bool`
- A **relation** is a term of type `'a ⇒ 'a ⇒ bool`
- Extensionality is built in: two sets with the same members are equal

Mathematics	HOL
$a \in S$	`S a`
$\{x \mid \varphi(x)\}$	`λx. φ x`
$S \subseteq T$	`∀x. S x ⟶ T x`
$S \cup T$	`λx. S x ∨ T x`
$a \to b$	`att a b`
for every set S	`∀S. …` with `S :: 'a ⇒ bool`

The whole encoding rests on this one idea. Extensions are predicates, semantics are predicates on predicates, and “for every extension” is a quantifier of the logic.

Automation: provers, model finders, Isabelle/HOL

- With Henkin semantics HOL has complete proof calculi: validity is **semi-decidable**
- **Automated theorem provers** search for a proof on their own: E and Vampire (first-order), Leo-III (higher-order), the SMT solvers Z3 and CVC4
- **Proof assistants** check proofs written by the user, step by step, with a small trusted kernel: Isabelle/HOL, Coq, Lean, HOL4
- Isabelle/HOL combines both, and adds a model finder

The Isabelle/HOL toolbox used in the library

- **Definitions**: `definition`, `abbreviation`, `type_synonym`, `datatype`. A defined symbol is a name for a HOL term
- **Isar**: structured, readable proofs
- **Sledgehammer**: hands the goal to external provers and SMT solvers, then rebuilds the proof inside Isabelle
- **Nitpick**: searches for finite models. A countermodel refutes a conjecture; with `satisfy` it produces a witness

Most proofs in the library are one-line Sledgehammer finds.
Most non-theorems are refuted by Nitpick in seconds.

Shallow semantical embeddings

- Right: modal logic K in HOL. A formula is a predicate on worlds
- **Shallow**: each object-level notion is an abbreviation that unfolds into plain HOL. No datatype of formulas, no evaluation function
- **Here**: not a logic with a syntax, but a **semantic theory**. Arguments are a type, sets are predicates
- Any HOL prover thereby becomes a reasoner for abstract argumentation[7]

Kripke Style Semantics (propositional modal logic K)

$M, g, s \models P$ if and only if $s \in g(P)$
$M, g, s \models \neg\, \varphi$ if and only if $M, g, s \not\models \varphi$
$M, g, s \models \varphi \vee \psi$ if and only if $M, g, s \models \varphi$ or $M, g, s \models \psi$
$M, g, s \models \Box\, \varphi$ if and only if for all t with sRt we have $M, g, t \models \varphi$

Standard Translation for Propositional Fragment (encoded in HOL)

- $\mathsf{P} = P_{i\to o}$
- $\neg = \lambda\varphi_{i\to o}\ \lambda w_i\ \neg(\varphi w)$
- $\vee = \lambda\varphi_{i\to o}\ \lambda\psi_{i\to o}\ \lambda w_i\ \varphi w \vee \psi w$
- $\Box = \lambda\varphi_{i\to o}\ \lambda w_i\ \forall v_i\ Rwv \to \varphi v$

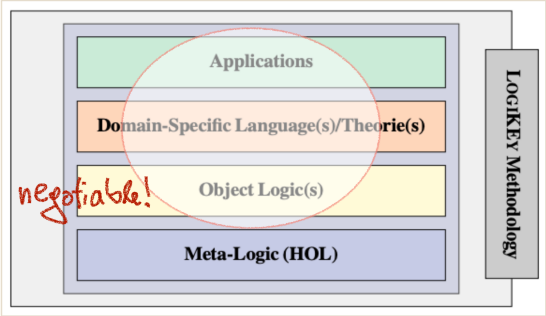
Validity

- $[\varphi_{i\to o}] = \forall w_i\ \varphi\ w$

[7] Benzmüller, *Universal (meta-)logical reasoning: recent successes*, Science of Computer Programming 2019.

Where it sits: the LogiKEy methodology

- **LogiKEy**:[8] HOL as a **universal meta-logic** for designing and testing normative and legal theories
- Four layers: **L0** the meta-logic HOL, **L1** object logics, **L2** domain theories, **L3** applications
- The argumentation library is an **L1 component**: domain theories and case studies are written on top of its predicates
- Experimental by design: definitions, lemmas and examples evolve together



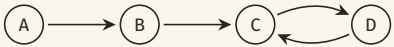
[8] Benzmüller et al., *Designing normative theories for ethical and legal reasoning: LogiKEy framework, methodology, and tool support*, Artificial Intelligence 2020.

03

The encoding: building blocks

First design decision: a framework is its attack relation

- The framework is **just the attack relation**, a term of type `'a Rel`
- The arguments are **all inhabitants of the type** `'a`. No carrier set, no pair
- Polymorphic: any type can play the role of `'a`. For the running example it is a datatype with the four values A, B, C, D
- The theories in `simplified/` take this literally and call the relation AF
- A semantics is then a predicate on a relation and a set
- Simple, and enough for abstract frameworks. Its limit is the subject of part 5




```
type_synonym 'a Rel = ‹'a ⇒ 'a ⇒ bool›

att :: 'a Rel       (* att a b: a attacks b *)
S   :: 'a Set       (* a set of arguments *)

conflictfreeExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
```

Sets and relations

```
type_synonym 'a Set = ‹'a ⇒ bool›          (* sets as functions into a 2-element codomain *)
type_synonym 'a Rel = ‹'a ⇒ 'a ⇒ bool›     (* relations as curried 2-ary functions *)

A ∪ B ≡ λx. (A x ∨ B x)      A ∩ B ≡ λx. (A x ∧ B x)      −A ≡ λx. ¬A x
A ⊆ B ≡ ∀x. (A x ⟶ B x)      A ≈ B ≡ ∀x. (A x ⟷ B x)

{|-|} ≡ λx. False      {|a|} ≡ λx. x = a      {|a,b|} ≡ λx. x = a ∨ x = b
```

- Union, intersection and complement are λ-terms over the connectives; inclusion and equality are quantified formulas
- ≈ is equality of sets by their members
- {|-|} is the empty set, {|a|} and {|a , b|} are finite sets written as predicates
- All of these are **abbreviations**. They unfold away, and the provers see plain HOL

Source: `misc.thy`. The same file holds the relative versions used in part 5.

Minimal, maximal, least, greatest

```
definition ‹minimal  Prop Obj E ≡ Prop Obj ∧ (∀X. Prop X ∧ E(X) ⊆ E(Obj) ⟶ E(X) ≈ E(Obj))›
definition ‹maximal  Prop Obj E ≡ Prop Obj ∧ (∀X. Prop X ∧ E(Obj) ⊆ E(X) ⟶ E(X) ≈ E(Obj))›
definition ‹least    Prop Obj E ≡ Prop Obj ∧ (∀X. Prop X ⟶ E(Obj) ⊆ E(X))›
definition ‹greatest Prop Obj E ≡ Prop Obj ∧ (∀X. Prop X ⟶ E(X) ⊆ E(Obj))›
```

`Prop`: **the candidates**

The property an object must have to take part, for example “is a complete extension of this framework”.

`Obj`: **the object**

The candidate that is claimed to be minimal, maximal, least or greatest. A set, a labelling, anything.

`E`: **the projection**

Turns an object into the set that is compared by inclusion. Written φ in the paper.

- **Minimal**: no candidate lies strictly below. **Least**: below every candidate. “Maximal” is about inclusion, never about the number of elements
- ∀X ranges over all objects of the given type, for instance all sets: **a higher-order quantifier**

Why the projection? One definition, nine uses

Semantics	**Notion**	**Candidates** `Prop`	**Projection** `E`
grounded extension	`minimal`	complete extensions	`id`
preferred extension	`maximal`	complete extensions	`id`
ideal extension	`greatest`	ideal sets	`id`
semi-stable extension	`maximal`	complete extensions	`range AF`
stage extension	`maximal`	conflict-free sets	`range AF`
grounded labelling	`minimal`	complete labellings	`in`
preferred labelling	`maximal`	complete labellings	`in`
semi-stable labelling	`minimal`	complete labellings	`undec`
stage labelling	`minimal`	conflict-free labellings	`undec`

- For extensions the projection is usually the identity: the sets themselves are compared
- Labellings are not sets. The projection picks the set to compare: the arguments labelled In, or those labelled Undec

Facts about the toolkit

```
✓ lemma leastMin: ‹least Prop Obj E ⟶ minimal Prop Obj E›
✗ lemma ‹minimal Prop Obj E ⟶ least Prop Obj E› nitpick oops            (* countermodel found *)
✓ lemma leastUnique: ‹least Prop A E ∧ least Prop B E ⟶ E(A) ≈ E(B)›
✓ lemma minLeastCollapse: ‹∃A. least Prop A E ⟹ ∀B. minimal Prop B E ⟶ least Prop B E›

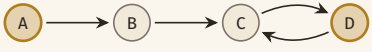
definition ‹MONO F ≡ ∀A B. A ⊆ B ⟶ F A ⊆ F B›
definition fixpoint :: ‹('a Set ⇒ 'a Set) ⇒ 'a Set ⇒ bool› where ‹fixpoint φ ≡ λS. φ(S) ≈ S›

✓ lemma wKTl: ‹MONO F ⟹ ∃S. least (fixpoint F) S id›          (* a weak Knaster-Tarski theorem *)
✓ lemma wKTg: ‹MONO F ⟹ ∃S. greatest (fixpoint F) S id›
```

- Least implies minimal, not conversely. A least element is unique; when one exists, every minimal element is least. The same holds for greatest and maximal
- `oops` closes a statement without proving it. After a `nitpick` call that found a countermodel it marks a **refuted** conjecture, kept in the file on purpose
- Every monotone function on sets has a least and a greatest fixed point: the least one is the intersection of all X with $F(X) \subseteq X$. This is the tool behind the grounded extension

Attacked and attacking sets, defence

- S^+: the arguments attacked by S. S^-: the arguments attacking S
- S **defends** a if every attacker of a is attacked by some member of S. Dung calls this acceptability
- The definition is the sentence, read word by word




```
definition plusset  :: ‹'a Rel ⇒ 'a Set ⇒ 'a Set› where ‹[att|S]⁺ ≡ λb. ∃a. S a ∧ att a b›
definition minusset :: ‹'a Rel ⇒ 'a Set ⇒ 'a Set› where ‹[att|S]⁻ ≡ λb. ∃a. S a ∧ att b a›

definition defends :: ‹'a Rel ⇒ 'a Set ⇒ 'a ⇒ bool›
  where ‹defends ≡ λatt S a. ∀b. att b a ⟶ (∃z. S z ∧ att z b)›

lemma defends_defEq: ‹defends att S a = [att|{|a|}]⁻ ⊆ [att|S]⁺›
  by (simp add: defends_def minusset_def plusset_def)
```

The lemma restates defence with the two set operations: the attackers of a are among the arguments attacked by S. Source: `base.thy`.

The characteristic function is `defends`

Dung's characteristic function maps a set S to the set $\mathcal{F}(S)$ of the arguments that S defends. In the library it needs no definition of its own:

```
abbreviation charFun :: ‹'a Rel ⇒ 'a Set ⇒ 'a Set› ("ℱ") where ‹ℱ ≡ defends›
```

Term	Type	Reading
`defends att S a`	`bool`	S defends a
`defends att S`	`'a ⇒ bool`, which is `'a Set`	the set of arguments defended by S, that is $\mathcal{F}(S)$
`defends att`	`'a Set ⇒ 'a Set`	the characteristic function of the framework

- Sets are predicates, and functions are applied one argument at a time. Leaving out the last argument of `defends` **turns a statement into a set**
- Verified in `base.thy`: the characteristic function is monotone (`ℱ_mono`), so it has a least and a greatest fixed point (`ℱ_leastFP_ex`, `ℱ_greatestFP_ex`)
- The least fixed point is Dung's grounded extension

Labellings: sets with three values

A set of arguments

```
'a Set = 'a ⇒ bool
```

Each argument is mapped to one of two values: inside or outside.

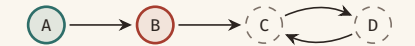
A labelling

```
'a Labelling = 'a ⇒ Label
```

Each argument is mapped to one of three values: **In**, **Out** or **Undec**.

```
datatype Label = In | Out | Undec
type_synonym 'a Labelling = ‹'a ⇒ Label›

in(Lab)    ≡ λx. Lab(x) = In
out(Lab)   ≡ λx. Lab(x) = Out
undec(Lab) ≡ λx. Lab(x) = Undec
```

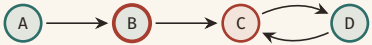


`in(Lab) = {|A|}` `out(Lab) = {|B|}` `undec(Lab) = {|C,D|}`

- `in`, `out` and `undec` turn a labelling back into three sets
- They are the **projections** of the previous slides: “the in-set is maximal” is `maximal ... Lab in`

Legally in, legally out, legally undecided

- **As-is state**: `in Lab x` says that the labelling labels x as In
- **Target state**: `legallyIn AF Lab x` says that x deserves the label In, given the labels of its attackers
- A labelling semantics states how the two must agree



B is labelled In, but its attacker A is In:

B is legally out, not legally in

```
(* An argument is termed legally-in if all of its attackers are labelled out *)
definition legallyIn :: ‹'a Rel ⇒ 'a Labelling ⇒ 'a ⇒ bool›
  where ‹legallyIn AF Lab ≡ λx. (∀y. (AF y x) ⟶ out Lab y)›
(* An argument is termed legally-out if it has an in-labelled attacker *)
definition legallyOut :: ‹'a Rel ⇒ 'a Labelling ⇒ 'a ⇒ bool›
  where ‹legallyOut AF Lab ≡ λx. (∃y. (AF y x) ∧ in Lab y)›
(* An argument is termed legally-undec if it is neither legally-in nor legally-out *)
definition legallyUndec :: ‹'a Rel ⇒ 'a Labelling ⇒ 'a ⇒ bool›
  where ‹legallyUndec AF Lab ≡ λx. ¬legallyIn AF Lab x ∧ ¬legallyOut AF Lab x›
```

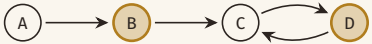
Source: `simplified/labellings-simpl.thy`.

04

The semantics as HOL predicates

Conflict-free and admissible extensions

- *S* is **conflict-free** if it does not contain two arguments such that one attacks the other
- *S* is **admissible** if it is conflict-free and defends each of its members



`conflictfreeExt att {|B,D|} holds,`
`admissibleExt att {|B,D|} does not`

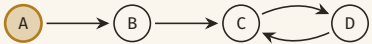
```
definition conflictfreeExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹conflictfreeExt ≡ λAF S. ∀a b. (S a ∧ S b) ⟶ ¬(AF a b)›

definition admissibleExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹admissibleExt ≡ λAF S. conflictfreeExt AF S ∧ (∀a. S a ⟶ defends AF S a)›
```

- An extension semantics is a predicate of type `'a Rel ⇒ 'a Set ⇒ bool`: **it holds of a framework and a set** exactly when the set is an extension
- Taking `a = b` in the first definition rules out self-attacking members

Complete extensions

S is a **complete extension** if it is admissible and contains each argument that it defends.



{A} defends A and nothing else: complete

```
definition completeExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹completeExt ≡ λAF S. admissibleExt AF S ∧ (∀a. defends AF S a ⟶ S a)›

(* Alternatively: conflict-free fixed points of ℱ, see [Dung 1995] Lemma 24 *)
definition completeExt2 :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹completeExt2 ≡ λAF S. conflictfreeExt AF S ∧ fixpoint (ℱ AF) S›
```

- Admissible says $S \subseteq \mathcal{F}(S)$; the second conjunct says $\mathcal{F}(S) \subseteq S$. Together: S is a fixed point of the characteristic function
- The library keeps **alternative definitions from the literature** side by side and proves them equivalent, here in `completeExt_defEq`
- Each such equivalence is a small test of the encoding

Preferred and grounded extensions

A **preferred extension** is a maximal complete extension, a **grounded extension** a minimal one, with respect to set inclusion.

```
definition preferredExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹preferredExt ≡ λAF S. maximal (completeExt AF) S id›

definition groundedExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹groundedExt AF S ≡ minimal (completeExt AF) S id›
```

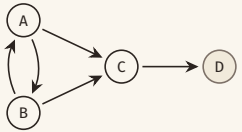
Unfolding `maximal` shows what the first definition says:

```
preferredExt AF S  =  completeExt AF S ∧ (∀E. completeExt AF E ∧ S ⊆ E ⟶ E ≈ S)
```

- E ranges over all predicates of type `'a ⇒ bool`: **a quantifier over sets inside the definition**. This is the step that propositional and first-order encodings cannot take directly
- Proved equivalent alternatives: preferred as maximal admissible (`preferredExt_defEq`), grounded as least complete and as least fixed point of $\mathcal{F}$ (`groundedExt_defEq1`, `groundedExt_defEq2`)

Ideal extensions

- An **ideal set** is an admissible set that is contained in every preferred extension
- The **ideal extension** is the greatest ideal set



Preferred: $\{A, D\}$ and $\{B, D\}$. Their intersection $\{D\}$ is not admissible, so the ideal extension is $\emptyset$

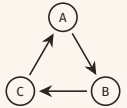
```
definition idealSet :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹idealSet ≡ λAF S. admissibleExt AF S ∧ (∀E. preferredExt AF E ⟶ S ⊆ E)›

definition idealExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹idealExt ≡ λAF S. greatest (idealSet AF) S id›
```

- Two set quantifiers are nested: one over the preferred extensions, and one inside `preferredExt` itself
- Here `greatest` is the right notion: ideal sets are closed under union (`idealSetUnion`), so the maximal one is the greatest (`idealExt_defEq`)

Stable, semi-stable and stage extensions

- **Stable**: conflict-free, and the range is the whole universe
- **Semi-stable**: complete, with maximal range among the complete extensions
- **Stage**: conflict-free, with maximal range among the conflict-free sets



Stable: none. Semi-stable: Ø.

Stage: {*A*}, {*B*}, {*C*}

```
definition range :: ‹'a Rel ⇒ 'a Set ⇒ 'a Set›
  where ‹range AF ≡ λS. S ∪ [AF|S]⁺›

definition stableExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹stableExt ≡ λAF S. conflictfreeExt AF S ∧ range AF S ≈ (λx. True)›
definition semistableExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹semistableExt ≡ λAF S. maximal (completeExt AF) S (range AF)›
definition stageExt :: ‹'a Rel ⇒ 'a Set ⇒ bool›
  where ‹stageExt ≡ λAF S. maximal (conflictfreeExt AF) S (range AF)›
```

- `(λx. True)` is the set of all arguments: in the simplified encoding, the whole type
- The projection is `range AF`: two sets are compared by their ranges, not by themselves

Conflict-free and admissible labellings

- A labelling is **conflict-free** if every In-labelled argument is not legally out, and every Out-labelled argument is legally out
- A labelling is **admissible** if every In-labelled argument is legally in, and every Out-labelled argument is legally out

```
definition conflictfreeLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹conflictfreeLab ≡ λAF Lab. ∀x. (in Lab x ⟶ ¬legallyOut AF Lab x) ∧ (out Lab x ⟶ legallyOut AF Lab x)›
definition admissibleLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹admissibleLab ≡ λAF Lab. ∀x. (in Lab x ⟶ legallyIn AF Lab x) ∧ (out Lab x ⟶ legallyOut AF Lab x)›
```

What the tools say about these definitions (`lab-properties.thy`):

✓ An admissible labelling always exists: label every argument Undec (`exAdmissible`)

✓ Admissible labellings are conflict-free (`admissibleLabConflictfree`). ✗ Not conversely

✓ In an admissible labelling, a legally undecided argument is labelled Undec (`admissibleLabLegUndec`). ✗ Not conversely

✗ A legally in argument need not be labelled In: admissibility puts no condition on the Undec label

Complete labellings

A labelling is **complete** if it is admissible and every Undec-labelled argument is legally undecided.

```
definition completeLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹completeLab ≡ λAF Lab. admissibleLab AF Lab ∧ (∀x. undec Lab x ⟶ legallyUndec AF Lab x)›
(* Alternative characterisation, [BCG 2011] Prop. 2 *)
definition completeLab2 :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹completeLab2 ≡ λAF Lab. (∀x. (in Lab x ⟷ legallyIn AF Lab x) ∧ (out Lab x ⟷ legallyOut AF Lab x))›
lemma complete_defEq: ‹completeLab AF Lab = completeLab2 AF Lab›
```

- Complete: **as-is and target coincide**. An argument is In exactly when it is legally in, Out exactly when it is legally out

✓ Legally in arguments are now labelled In, legally out ones Out (`completeLabLegIn`, `completeLabLegOut`)

✓ The in-set is a fixed point of the characteristic function (`completeLabInFP`)

✓ The in-set determines a complete labelling, and so does the out-set (`completeLabInLab`, `completeLabOutLab`)

✗ The undec-set does not: Nitpick finds two complete labellings with the same undecided arguments

Preferred, grounded and ideal labellings

```
definition preferredLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹preferredLab AF Lab ≡ maximal (completeLab AF) Lab in›
definition groundedLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹groundedLab AF Lab ≡ minimal (completeLab AF) Lab in›
definition lessOrEquallyCommittedLab :: ‹'a Labelling ⇒ 'a Labelling ⇒ bool› (infix "⊑" 52)
  where "L1 ⊑ L2 ≡ (in(L1) ⊆ in(L2)) ∧ (out(L1) ⊆ out(L2))"
definition qidealLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹qidealLab AF Lab ≡ admissibleLab AF Lab ∧ (∀L. preferredLab AF L ⟶ Lab ⊑ L)›
definition idealLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹idealLab AF Lab ≡ qidealLab AF Lab ∧ (∀L. qidealLab AF L ⟶ L ⊑ Lab)›
```

- Preferred and grounded: the same two lines as for extensions, with the projection `in` in place of `id`
- Ideal needs an order on labellings: L1 $\sqsubseteq$ L2, “L1 is less or equally committed than L2”, compares in-sets and out-sets
- **Quasi-ideal** labellings play the role of ideal sets; the ideal labelling is the greatest of them with respect to $\sqsubseteq$
- Here the quantifiers range over labellings, that is, over functions

Stable, semi-stable and stage labellings

```
definition stableLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹stableLab AF Lab ≡ completeLab AF Lab ∧ (∀x. Lab(x) ≠ Undec)›

definition semistableLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹semistableLab AF Lab ≡ minimal (completeLab AF) Lab undec›

definition stageLab :: ‹'a Rel ⇒ 'a Labelling ⇒ bool›
  where ‹stageLab AF Lab ≡ minimal (conflictfreeLab AF) Lab undec›
```

Extensions: make the range large

Stable: the range is everything.

Semi-stable and stage: the range $S \cup S^+$ is maximal.

Labellings: make the undecided part small

Stable: nothing is Undec.

Semi-stable and stage: the undec-set is minimal.

- The two views match: the range of an extension is what its labelling marks In or Out; the rest is Undec
- The projection changes from `range` AF to `undec`, and `maximal` becomes `minimal`

The whole encoding on one slide

Semantics	A set S is an extension if …	A labelling is one if …
conflict-free	no member attacks a member	In ⇒ not legally out; Out ⇒ legally out
admissible	conflict-free, and S defends its members	In ⇒ legally in; Out ⇒ legally out
complete	admissible, and S contains all it defends	admissible, and Undec ⇒ legally undecided
grounded	`minimal` complete, `id`	`minimal` complete, `in`
preferred	`maximal` complete, `id`	`maximal` complete, `in`
ideal	`greatest` ideal set, `id`	greatest quasi-ideal labelling, $\sqsubseteq$
stable	conflict-free, and the range is everything	complete, and no argument is Undec
semi-stable	`maximal` complete, `range`	`minimal` complete, `undec`
stage	`maximal` conflict-free, `range`	`minimal` conflict-free, `undec`

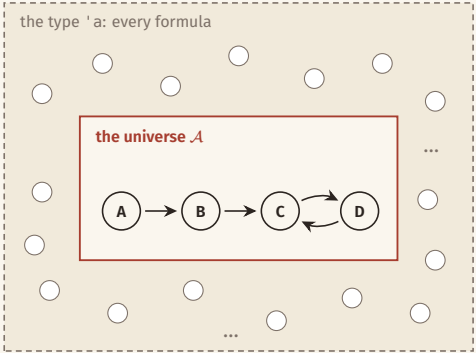
- **Eighteen predicates from four ingredients**: defence, the three “legally” notions, inclusion, and one generic notion of minimal and maximal with a projection
- Each side takes a few dozen lines of Isabelle

05

Second pass: an explicit universe of arguments

What goes wrong with “all inhabitants of the type”

- Suppose the arguments are to be **formulas**: instantiate `'a` with a type PROP of propositional formulas
- PROP has infinitely many inhabitants, and all of them count as arguments
- So **every framework over PROP is infinite**, and there is no way to say “the framework consists of these five formulas”
- The same happens whenever a framework is a **part** of something larger: the arguments held by one agent, a graph after an argument is removed
- The type is fixed once and for all. A set of arguments can vary



We want to talk about $\mathcal{A}$. The type gives us everything.

The fix: an explicit universe of arguments

- A framework is now $(\mathcal{A}, att)$, with $\mathcal{A}$ `:: 'a Set` the set of its arguments. The two are passed as separate parameters, not as a pair
- Every notion takes the universe as an extra first parameter, written as a superscript
- Quantifiers are restricted to the universe, as in free logic: “for all” gets a guard, “there is” gets a conjunct

```
A ⊆ᵁ B ≡ ∀x. U x ⟶ (A x ⟶ B x)              (* inclusion, as far as U can see *)
A ≈ᵁ B ≡ ∀x. U x ⟶ (A x ⟷ B x)              (* equality,  as far as U can see *)

∀ᵁ P ≡ ∀x. U x ⟶ (P x)                      (* relative quantifiers *)
∃ᵁ P ≡ ∃x. U x  ∧  (P x)

lemma relQuantifierDual: ‹∀^𝒜 P ⟷ ¬(∃^𝒜 (λx. ¬(P x)))› by simp
```

- In a dependently typed system the restriction could live in the types. In simply typed HOL it has to **live in the terms**
- The gain: sub-frameworks, and frameworks of structured arguments, are **values of one type**

Source: `misc.thy`, where the universe is called U; from `base.thy` on it is called $\mathcal{A}$.

The building blocks, relative to the universe

```
(* misc.thy: orders and fixed points *)
minimal^U Prop Obj E ≡ Prop Obj ∧ (∀X. Prop X ∧ (E X) ⊆^U (E Obj) ⟶ (E X) ≈^U (E Obj))
MONO^U F ≡ ∀A B. (A ⊆^U B ⟶ F A ⊆^U F B)                fixpoint^U φ ≡ λS. φ(S) ≈^U S
lemma wKTl_relw: ‹MONO^U F ⟹ ∃S. least^U (fixpoint^U F) S id›

(* base.thy: attack images and defence *)
[𝒜|att|S]^+ ≡ λb. (∃a. 𝒜 a ∧ S a ∧ att a b)
defends^𝒜 ≡ λatt S a. ∀b. 𝒜 b ⟶ att b a ⟶ (∃z. 𝒜 z ∧ S z ∧ att z b)
lemma defends_rel_defEq: ‹𝒜 a ⟹ (defends^𝒜 att S a) = [𝒜|att|{|a|}]⁻ ⊆^𝒜 [𝒜|att|S]^+›
abbreviation charFun_rel :: ‹'a Set ⇒ 'a Rel ⇒ 'a Set ⇒ 'a Set› ("ℱ^𝒜") where ‹ℱ^𝒜 ≡ defends^𝒜›

(* labellings.thy *)
legallyIn^𝒜 att Lab  ≡ λx. (∀y. 𝒜 y ⟶ (att y x) ⟶ out Lab y)
legallyOut^𝒜 att Lab ≡ λx. (∃y. 𝒜 y ∧ (att y x) ∧ in Lab y)
```

- Nothing new happens: each definition of part 3 is repeated with the universe threaded through
- Only attackers and defenders inside $\mathcal{A}$ count. What `att` says about other inhabitants of the type is ignored
- The Knaster-Tarski lemma and the properties of the characteristic function are proved again, in relative form

Simplified and relative, side by side

`simplified/`: the universe is the type

```
defends ≡ λatt S a.
  ∀b. att b a ⟶ (∃z. S z ∧ att z b)

conflictfreeExt ≡ λAF S.
  ∀a b. (S a ∧ S b) ⟶ ¬(AF a b)

admissibleExt ≡ λAF S. conflictfreeExt AF S ∧
  (∀a. S a ⟶ defends AF S a)

completeExt ≡ λAF S. admissibleExt AF S ∧
  (∀a. defends AF S a ⟶ S a)

preferredExt ≡ λAF S.
  maximal (completeExt AF) S id
```

Main theories: the universe is a parameter

```
defends^𝒜 ≡ λatt S a.
  ∀b. 𝒜 b ⟶ att b a ⟶ (∃z. 𝒜 z ∧ S z ∧ att z b)

conflictfreeExt^𝒜 ≡ λatt S.
  ∀a b. (𝒜 a ∧ 𝒜 b) ⟶ (S a ∧ S b) ⟶ ¬(att a b)

admissibleExt^𝒜 ≡ λatt S. conflictfreeExt^𝒜 att S ∧
  (∀a. 𝒜 a ⟶ S a ⟶ defends^𝒜 att S a)

completeExt^𝒜 ≡ λatt S. admissibleExt^𝒜 att S ∧
  (∀a. 𝒜 a ⟶ defends^𝒜 att S a ⟶ S a)

preferredExt^𝒜 ≡ λatt S.
  maximal^𝒜 (completeExt^𝒜 att) S id
```

- Rule of thumb: **remove the highlighted guards and the superscripts**, and the relative definition becomes the simplified one
- With the whole type as universe the guards are trivially true
- From here on the slides show the main theories

What the explicit universe costs

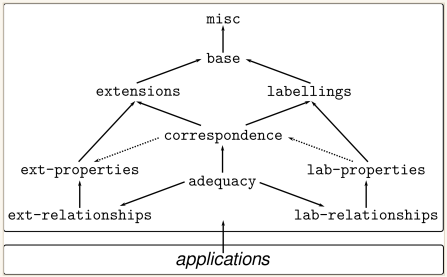
- Predicates are **unconstrained outside the universe**. Two predicates that agree on $\mathcal{A}$ describe the same extension, even if they differ elsewhere
- So sets are compared with $\approx^{\mathcal{A}}$, never with =, and labellings with $\cong^{\mathcal{A}}$
- Statements about a single argument need the guard $\mathcal{A}$ a

```
✓ lemma subs_rel: ‹A ⊆ B ⟹ A ⊆ᵁ B› by simp
✗ lemma ‹A ⊆ᵁ B ⟶ A ⊆ B› nitpick oops
✗ lemma ‹minimal Prop Obj E ⟶ minimalᵁ Prop Obj E› nitpick oops
✗ lemma ‹minimalᵁ Prop Obj E ⟶ minimal Prop Obj E› nitpick oops
✓ lemma least_rel_impl: ‹least Prop Obj E ⟶ leastᵁ Prop Obj E›

✓ lemma equivLabId: "L1 ≅ L2 ⟷ L1 = L2"
✗ lemma equivLabId_rel: "L1 ≅^𝒜 L2 ⟷ L1 = L2" oops    (* does not hold as expected *)
✓ lemma IdLabImplEquiv_rel: "L1 = L2 ⟹ L1 ≅^𝒜 L2"
```

- Global inclusion implies relative inclusion, not conversely. Minimality transfers in neither direction
- The price is paid once, in the foundations; the definitions of the semantics stay as short as before

Structure of the session



Theory graph of the library. Solid arrows: definitions are reused; dotted arrows: lemmas are used in proofs. Figure: Steen & Fuenmayor, J. Logic and Computation 34(2), 2024.

```
AA_Fixed/
  misc.thy                sets, relations, orderings, fixed points
  Zorn-lemma.thy          maximal elements, Zorn's lemma
  base.thy                attack images, defence, labels, acceptance
  extensions.thy          extension-based semantics
  labellings.thy          labelling-based semantics
  ext-properties.thy      properties of extensions
  lab-properties.thy      properties of labellings
  ext-relationships.thy   inclusions among extension semantics
  lab-relationships.thy   inclusions among labelling semantics
  correspondence.thy      extensions and labellings
  adequacy.thy            exemplary theorems of the paper
  regression-checks.thy   regression theorems
  simplified/             the same, over the whole type
    model-generation.thy  generation with Nitpick
  ROOT                    one session, 20 theories
```

- Parts 3 and 4 of this talk: `misc`, `base`, `extensions`, `labellings`
- Part 6: the properties, relationships, correspondence and adequacy theories
- Part 7: `simplified/model-generation`

06

Reasoning about argumentation: the meta-theory

What does “adequate” mean here?

- The encoding **restates** the definitions of abstract argumentation in HOL, clause by clause
- There is no separate object syntax whose translation could be proved faithful once and for all. The papers leave a general faithfulness theorem as future work
- Adequacy is therefore argued **from the inside**: the encoded notions must behave as the literature says they do
- Each item on the right is a group of statements in the session, proved or refuted by the tools

The evidence

- ✓ The **textbook results** are provable: lemmas and theorems of Dung 1995, propositions of BCG 2011
- ✓ **Alternative definitions** from the literature are equivalent: the `_defEq` lemmas
- ✓ The **inclusions** between the semantics hold
- ✓ **Extensions and labellings** correspond
- ✗ The known **non-theorems** have countermodels
- ✓ The **benchmark graphs** give the expected answers

First properties

```
(* no self-attacks in a conflict-free extension, [BCG 2011] p. 8 *)
lemma cf_noselfattacks: ‹conflictfreeExt⇘A⇙ att S ⟹ ¬ (∃⇗A⇖ (λa. (S a) ∧ (att a a)))›
  by (simp add: conflictfreeExt_def)

(* the characteristic function preserves conflict-freeness *)
lemma cf_preserveChar: ‹conflictfreeExt⇘A⇙ att S ⟹ conflictfreeExt⇘A⇙ att (ℱ⇗A⇖ att S)›
  by (smt (verit) conflictfreeExt_def defends_rel_def)

(* a conflict-free set S is admissible iff it is a subset of ℱ(S), [Dung 1995] Lemma 18 *)
lemma cf_admissibleChar: ‹conflictfreeExt⇘A⇙ att S ⟹ admissibleExt⇘A⇙ att S ⟷ S ⊆⇗A⇖ ℱ⇗A⇖ att S›
  by (simp add: admissibleExt_def)
```

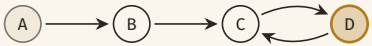
- The statements are the textbook ones, written with the predicates of parts 3 to 5. Free variables are implicitly universally quantified: **for every** universe, attack relation and set
- The proofs are **one line each**: unfold the definitions, then call the simplifier or an SMT solver. Sledgehammer found them
- `smt (verit)` means: the SMT solver veriT found a proof, and Isabelle replayed it step by step in its own kernel

Source: `adequacy.thy`, which follows the numbering of the lemmas in the paper.

Dung's Fundamental Lemma

Fundamental Lemma (Dung 1995, Lemma 10). Let S be admissible, and let S defend a and a'. Then

1. $S \cup \{a\}$ is admissible, and
2. $S \cup \{a\}$ defends a'.



$S = \{D\}$ is admissible and defends A,

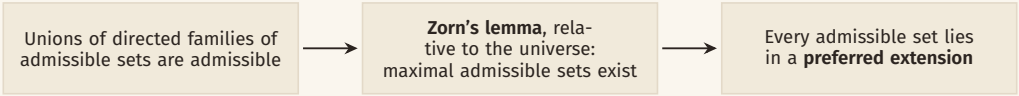
so $\{A, D\}$ is admissible

```
lemma dung_fundlemma_1:
  ‹(admissibleExt⇘𝒜⇙ att S) ∧ (defends⇘𝒜⇙ att S a) ⟹ admissibleExt⇘𝒜⇙ att (S ∪ {a})›
  by (smt (z3) admissibleExt_def conflictfreeExt_def defends_rel_def)

lemma dung_fundlemma_2:
  ‹(admissibleExt⇘𝒜⇙ att S) ∧ (defends⇘𝒜⇙ att S a) ∧ (defends⇘𝒜⇙ att S a') ⟹ defends⇘𝒜⇙ att (S ∪ {a}) a'›
  by (metis (full_types, lifting) MONO_def ℱ_mono')
```

- Admissible sets can be extended with the arguments they defend. This is the engine behind Dung's existence results
- The second part needs only that the characteristic function is monotone

From admissible sets to preferred extensions




```
(* admissible sets form a directed-complete partial order; [Dung 1995] Theorem 11(1) has ω-complete *)
lemma admissibleDirectedComplete: "dcpo.𝒜 (admissibleExt.𝒜 att)"
lemma admissibleOmegaComplete: "ω-cpo.𝒜 (admissibleExt.𝒜 att)"

lemma ZornLemma2_ω_rel: "ω-cpo.U A ⟹ ∀X. A X ⟶ (∃M. maximal.U A M id ∧ X ⊆.U M)"

(* for each admissible set there is a preferred extension extending it, [Dung 1995] Theorem 11(2) *)
lemma DungTh11: "admissibleExt.𝒜 att S ⟶ (∃E. S ⊆.𝒜 E ∧ preferredExt.𝒜 att E)"
```

- Dung shows that the admissible sets form an ω-complete partial order. The library proves more: they are **directed complete**
- Zorn's lemma is itself proved in the session (`Zorn-lemma.thy`), first globally, then relative to a universe
- **No finiteness assumption**: the statements hold for every universe $\mathcal{A}$, finite or not

Existence and uniqueness

```
lemma completeExtExists:  ‹∃E. completeExt⇘𝒜⇙ att E›
lemma preferredExtExists: ‹∃E. preferredExt⇘𝒜⇙ att E›
lemma groundedExtExists:  ‹∃E. groundedExt⇘𝒜⇙ att E›
(* the grounded extension is the least complete one, hence unique: [BCG 2011] Prop. 4 *)
lemma ‹groundedExt⇘𝒜⇙ att E ⟷ least⇘𝒜⇙ (completeExt⇘𝒜⇙ att) E id›
(* conflict-free sets are complete iff they are fixed points of ℱ: [Dung 1995] Lemma 24 *)
lemma complete_fix: ‹conflictfreeExt⇘𝒜⇙ att S ⟹ completeExt⇘𝒜⇙ att S ⟷ S ≈⇘𝒜⇙ ℱ⇘𝒜⇙ att S›
lemma completeLabExists: ‹∃Lab. completeLab⇘𝒜⇙ att Lab›     (* also groundedLabExists, preferredLabExists *)
lemma grounded_unique: ‹groundedLab⇘𝒜⇙ att Lab1 ⟹ groundedLab⇘𝒜⇙ att Lab2 ⟹ ∀x. 𝒜 x ⟶ Lab1(x) = Lab2(x)›
lemma semistableLabExistsFinite: assumes "finite {x. 𝒜 x}" shows "∃Lab. semistableLab⇘𝒜⇙ att Lab"
(* stable labellings do not exist necessarily *)
corollary ‹(∃x. 𝒜 x) ⟶ (∃att. ∀Lab. ¬stableLab⇘𝒜⇙ att Lab)›
```

- Complete, preferred and grounded extensions **always exist**, and so do the corresponding labellings. Semi-stable labellings exist in every finite framework
- The grounded extension is the least complete one and the least fixed point of the characteristic function: the three definitions of slide 29 agree, so it is unique
- Stable labellings **need not exist**: the relation in which every argument attacks itself has none

Properties of labellings

```
lemma complete_sets_mono:
  ‹completeLab⇘A⇙ att Lab1 ∧ completeLab⇘A⇙ att Lab2 ⟹ in(Lab1) ⊆⇘A⇙ in(Lab2) ⟷ out(Lab1) ⊆⇘A⇙ out(Lab2)›

(* grounded labellings: minimal in-set iff minimal out-set, [BCG 2011] Prop. 5 *)
lemma complete_min_inout: ‹minimal⇘A⇙ (completeLab⇘A⇙ att) Lab in ⟷ minimal⇘A⇙ (completeLab⇘A⇙ att) Lab out›

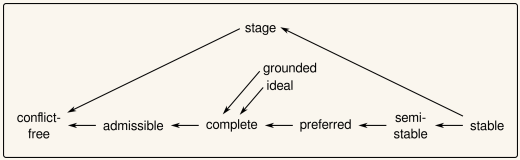
(* preferred labellings: maximal in-set iff maximal out-set, [BCG 2011] Prop. 8 *)
lemma preferred_max_inout: ‹maximal⇘A⇙ (completeLab⇘A⇙ att) Lab in ⟷ maximal⇘A⇙ (completeLab⇘A⇙ att) Lab out›

(* every stable labelling is preferred, after [Dung 1995] Lemma 15 *)
lemma lem15Dung: ‹stableLab⇘A⇙ att Lab ⟹ preferredLab⇘A⇙ att Lab›
```

- **More In means more Out**: among complete labellings, the in-sets and the out-sets are ordered in the same way
- So grounded and preferred labellings can be defined through either set. The projection `in` of slide 34 could as well be `out`
- Changing the projection is changing one word of the statement. The generic definition of slide 20 pays off here

Sources: `adequacy.thy`, `lab-properties.thy`.

The landscape, verified



- **Every arrow is a lemma**: nine for extensions, nine for labellings
- **No arrow can be reversed**: for each converse the model finder produces a countermodel

```
(* ext-relationships.thy: one lemma per arrow *)
stable_implies_semistable_ext     semistable_implies_preferred_ext     preferred_implies_complete_ext
complete_implies_admissible_ext   admissible_implies_cf_ext            grounded_implies_complete_ext
ideal_implies_complete_ext        stable_implies_stage_ext             stage_implies_cf_ext

lemma stable_implies_stage_ext: ‹stableExt⇩A AF S ⟹ stageExt⇩A AF S›
  by (simp add: greatestMax_rel greatest_rel_def stableExt_def stageExt_def)
```

- Most of these proofs only unfold a definition. Two need an SMT solver: semi-stable implies preferred, and stable implies semi-stable
- An example from the paper: “every stage labelling is stable” is refuted by Nitpick in less than a second
- The diagram of the survey has become a **checked** part of the theory

Extensions and labellings correspond

```
definition Lab2Ext :: ‹'a Labelling ⇒ 'a Set›
  where ‹Lab2Ext Lab ≡ in(Lab)›
definition Ext2Lab :: ‹'a Set ⇒ 'a Rel ⇒ 'a Set ⇒ 'a Labelling›
  where ‹Ext2Lab^𝒜 att E ≡ λa. if (E a) then In else (if ([𝒜|att|E]⁺ a) then Out else Undec)›

✓ lemma preferred_LE:  "preferredLab^𝒜 att Lab ⟶ preferredExt^𝒜 att (Lab2Ext Lab)"
✗ lemma preferred_LE': "preferredExt^𝒜 att (Lab2Ext Lab) ⟶ preferredLab^𝒜 att Lab" nitpick oops
✓ lemma preferred_EL:  "preferredExt^𝒜 att E ⟶ preferredLab^𝒜 att (Ext2Lab^𝒜 att E)"
✓ lemma preferred_EL': "preferredLab^𝒜 att (Ext2Lab^𝒜 att E) ⟶ preferredExt^𝒜 att E"
```

- From a labelling to an extension: take the in-set. This needs no framework
- From an extension to a labelling: the members are In, the arguments they attack are Out, the rest is Undec
- The same **family of four lemmas** is stated for all nine semantics, ideal included: _LE, _LE', _EL, _EL'
- Why `preferred_LE'` fails: the in-set alone does not tell which of the other arguments are Out and which are Undec. Only `Ext2Lab` picks the right labelling

Source: `correspondence.thy`.

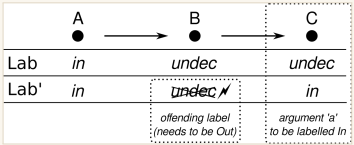
Theory exploration I: a wrong conjecture

- Is there a Fundamental Lemma for labellings? **First attempt**: if *Lab* is admissible and its in-set defends *a*, then relabelling *a* as In gives an admissible labelling again

```
lemma NaiveDungFundLemmaLab: ‹admissibleLab𝒜 att Lab ∧ defends𝒜 att (in Lab) a ⟶
                               admissibleLab𝒜 att ( λx. if (x = a) then In else (Lab x) )›
  nitpick
```

Proof state | Auto update | Update | Search: | 100%

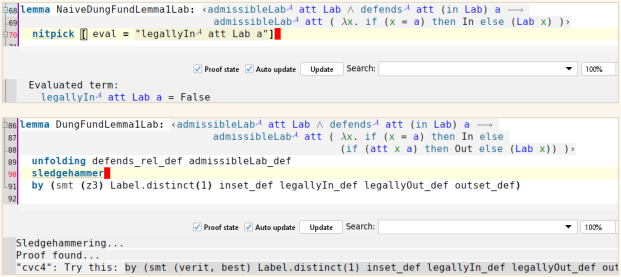
```
Nitpick found a counterexample for card 'a = 3:
  Free variables:
    Lab = (λx. _)(A := In, B := Undec, C := Undec)
    𝒜 = (λx. _)(A := True, B := True, C := True)
    a = C
    att =
      (λx. _)
       ((A, A) := False, (A, B) := True, (A, C) := False, (B, A) := False, (B, B) := False,
        (B, C) := True, (C, A) := False, (C, B) := False, (C, C) := False)
```



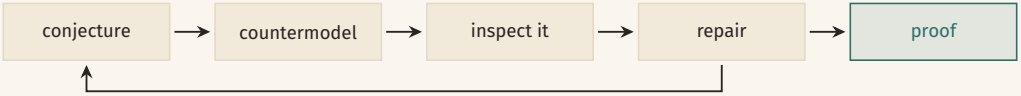
- Nitpick answers within a second with a framework of three arguments, A attacks B and B attacks C, the labelling (In, Undec, Undec), and *a* = C
- The labelling is admissible, and its in-set {*A*} defends C against B. After relabelling, C is In while its attacker B is still Undec

Screenshot and figure: A. Steen, D. Fuenmayor, J. Logic and Computation 34(2), 2024, Section 4.1. The naive lemma is in the paper only, not in the session.

Theory exploration II: repair and prove



- Top: ask Nitpick to **evaluate** a term in its countermodel. Is *a* legally in after relabelling? No
- **Repair**: also relabel the attackers of *a* as Out
- Bottom: Sledgehammer finds a proof; the line starting with by is checked by Isabelle
- This is lemma `DungFundLemma1Lab` of `lab-properties.thy`

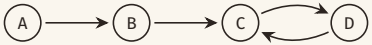


A new meta-theoretical result, found in dialogue with the tools.

07

Reasoning with argumentation: generating extensions and labellings

Encoding a concrete framework



- A `datatype` with four constructors: four distinct arguments, and no others
- The attack relation is defined by cases; the last line makes every other pair a non-attack
- The datatype **is** the universe, so the simplified encoding fits
- The two **exhaustion lemmas** turn a quantifier over `Arg` into four cases
- One locale per benchmark graph: `ExFig4`, `ExFig5`, `ExFig6`

```
(* Example set-up from [BG2011], Figure 4 *)
locale ExFig4 begin
datatype Arg = A | B | C | D
  fun att :: ‹Arg Rel› where
    "att A B = True" |
    "att B C = True" |
    "att C D = True" |
    "att D C = True" |
    "att _ _ = False"

lemma all_Arg: "(∀x::Arg. P x) ⟷ P A ∧ P B ∧ P C ∧ P D"
  by (metis Arg.exhaust)

lemma ex_Arg: "(∃x::Arg. P x) ⟷ P A ∨ P B ∨ P C ∨ P D"
  by (metis Arg.exhaust)
```

Source: `simplified/model-generation.thy`.

Asking Nitpick for a labelling

- State "*Lab* is a stable labelling of *att*" as if it were a lemma. Lab is a free variable
- `satisfy`: do not look for a countermodel, look for values of the free variables that make the statement **true**
- **The model is the labelling**: A and D are In, B and C are Out
- `oops`: the statement was a question, not a theorem. Nothing is asserted
- `card Arg=4`: search with exactly the four arguments

```
nitpick_params [box=false, card Label=3, sat_solver=SAT4J]

lemma ‹stableLab att Lab› nitpick[card Arg=4,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}
```



Query: the one of LNGAI 2021, Fig. B.2, written with the session's names and options.

Nitpick output: the same figure.

Enumerating all of them: `findFor'`

```
abbreviation ‹findFor' att Prop S ≡ ∀x. x ∈ S ⟷ Prop att x›

lemma ‹findFor' att preferredLab Labs› nitpick[card Arg=4,satisfy,box=false] oops
```

```
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)}
```

- `findFor' att Prop S` says that S is **exactly the set of objects** satisfying the semantics. With S free, a model must list all of them
- Here: the two preferred labellings of the running example, as in BCG

How many?	conflict-free	admissible	complete	grounded	preferred	stable	semi-stable	ideal
extensions	8	5	3	1	2	2	2	1
labellings	–	7	3	1	2	2	2	1

All numbers agree with BCG 2011. Five admissible sets but seven admissible labellings: an in-set does not fix the other labels. The ideal labelling is proved rather than searched (slide 57).

Flexibly constrained generation

```
abbreviation surjective where "surjective f ≡ ∀y. ∃x. f x = y"

(* Admissible labelling where A is In *)
✓ lemma ‹admissibleLab att Lab ∧ (Lab A) = In› nitpick[card Arg=4,satisfy] oops

(* Admissible labelling where Lab is surjective *)
✓ lemma ‹admissibleLab att Lab ∧ (surjective Lab)› nitpick[card Arg=4,satisfy] oops

(* More than two In arguments are impossible; prove the bound directly. *)
✓ lemma admissible_at_most_two_in:
    assumes adm: "admissibleLab att Lab"
    shows "card {x. in Lab x} ≤ 2"
```

✓ on a query: Nitpick finds a model. ✓ on the last lemma: proved.

- The constraint is **any HOL formula** about the labelling: a fixed label for one argument, a property of the function such as “all three labels are used”, a bound on a cardinality
- A dedicated solver has no input language for such questions
- A model finder settles **existence**. That no admissible labelling has three In arguments is a statement about all labellings: it is **proved**, by a case analysis in Isar

Source: `simplified/model-generation.thy`, locale `ExFig4`.

Proving instead of searching, and the limits

```
(* Kernel-checked witnesses complement the exploratory Nitpick queries. *)
lemma preferred_AC: "preferredExt att {A,C}"
  unfolding preferredExt_def maximal_def completeExt_def admissibleExt_def
    conflictfreeExt_def defends_def id_def
  by (auto simp: all_Arg ex_Arg)
lemma ideal_A: "idealExt att {A}"
lemma ideal_labelling: "idealLab att (λx. if x = A then In else if x = B then Out else Undec)"
lemma only_empty_admissible: "admissibleExt att S ⟷ S ≈ {}"          (* the odd cycle, locale ExFig6 *)

(* base.thy: skeptical and credulous justification of an argument under a labelling semantics S *)
sJ S att ≡ λarg. ∀Lab. S att Lab ⟶ Lab(arg) = In
cJ S att ≡ λarg. ∃Lab. (S att Lab) ∧ Lab(arg) = In
lemma skeptical_credulous_nonempty: "(∃Lab. S att Lab) ⟹ sJ S att a ⟹ cJ S att a"
```

- Unfold the definitions, replace the quantifiers by four cases, simplify: the kernel confirms what the model finder suggested
- An argument is skeptically justified if it is In in every labelling of the semantics, credulously if in some

Limits

- A Nitpick model speaks about one finite framework
- “No model found” is not a proof
- Nested maximality is slow: an ideal-extension query takes about a minute
- Not a competitor to SAT-based solvers on large graphs

08

Positioning, outlook, summary

Positioning: expressivity versus scalability

Dedicated solvers: SAT, QBF, answer-set programming

- Decidable target languages, efficient procedures
- Large frameworks
- A fixed menu of semantics and of queries
- They compute extensions; they do not reason about the semantics

The HOL embedding

- Semi-decision procedures; model finding on small finite frameworks
- Any property that can be written in HOL can be asked for or conjectured
- Object level and meta level in one language
- Proved statements are checked by a small kernel

A bridge between abstract argumentation and automated deduction:
it complements the solvers, it does not compete with them

- The encoding is not tied to one system: any prover for HOL can read the same definitions
- A good division of labour between provers, SMT solvers and model finders on larger inputs is engineering work that the papers leave open

Outlook

Further work named in the papers

- **More semantics**: eager, CF2, stage2, with their meta-theory
- **Richer frameworks**: joint and higher-order attacks, bipolar frameworks with a support relation
- **Instantiated arguments**: pairs (support, claim) of formulas of an embedded logic; rebutting, undermining and undercutting attacks derived from its consequence relation
- **Scaling**: the limits of the approach on larger inputs
- **Explanation semantics** and a combination with the LogiKEy normative theories

What the explicit universe makes possible

- A framework, a part of it and a modified copy are values of one type: one can quantify over them and compare their extensions
- The library can serve as the **object-logic layer** of a LogiKEy development, with domain theories and case studies on top
- Sessions built on `AA_Fixed` in this way: `Implicit_Disagreement` (agents, coalitions and their hidden disagreements) and `Jiminy` (a moral advisor)
- Structured argumentation in the same style: the session `ASPIC_Plus`

Summary

1. **Sets are predicates.** An extension is a term of type `'a ⇒ bool`, a labelling a term of type `'a ⇒ Label`
2. **Semantics are predicates on predicates.** Nine semantics, each for extensions and for labellings, from defence, legality and inclusion
3. **One notion of minimal and maximal**, with a projection, covers grounded, preferred, ideal, semi-stable and stage
4. **An explicit universe** makes room for instantiated arguments and sub-frameworks
5. **The textbook is verified**: Dung's lemmas, the inclusions, the correspondence; the non-theorems are refuted
6. **A model finder generates** extensions and labellings, under any constraint expressible in HOL

The session: `AA_Fixed`. The original library: `github.com/aureleeNet/formalizations`

Fuenmayor and Steen, *A Flexible Approach to Argumentation Framework Analysis using Theorem Proving*, LNGAI 2021

Steen and Fuenmayor, *A formalization of abstract argumentation in higher-order logic*, Journal of Logic and Computation 34(2) 2024

Backup: the labelling encoding in full

```
datatype Label = In | Out | Undec                type_synonym 'a Labelling = ‹'a ⇒ Label›
in(Lab) ≡ λx. Lab(x) = In            out(Lab) ≡ λx. Lab(x) = Out            undec(Lab) ≡ λx. Lab(x) = Undec

legallyIn    AF Lab ≡ λx. (∀y. (AF y x) ⟶ out Lab y)
legallyOut   AF Lab ≡ λx. (∃y. (AF y x) ∧ in Lab y)
legallyUndec AF Lab ≡ λx. ¬legallyIn AF Lab x ∧ ¬legallyOut AF Lab x

conflictfreeLab ≡ λAF Lab. ∀x. (in Lab x ⟶ ¬legallyOut AF Lab x) ∧ (out Lab x ⟶ legallyOut AF Lab x)
admissibleLab   ≡ λAF Lab. ∀x. (in Lab x ⟶ legallyIn AF Lab x) ∧ (out Lab x ⟶ legallyOut AF Lab x)
completeLab     ≡ λAF Lab. admissibleLab AF Lab ∧ (∀x. undec Lab x ⟶ legallyUndec AF Lab x)

groundedLab   AF Lab ≡ minimal (completeLab AF) Lab in
preferredLab  AF Lab ≡ maximal (completeLab AF) Lab in

L1 ⊑ L2 ≡ (in(L1) ⊆ in(L2)) ∧ (out(L1) ⊆ out(L2))
qidealLab     AF Lab ≡ admissibleLab AF Lab ∧ (∀L. preferredLab AF L ⟶ Lab ⊑ L)
idealLab      AF Lab ≡ qidealLab AF Lab ∧ (∀L. qidealLab AF L ⟶ L ⊑ Lab)

stableLab     AF Lab ≡ completeLab AF Lab ∧ (∀x. Lab(x) ≠ Undec)
semistableLab AF Lab ≡ minimal (completeLab AF) Lab undec
stageLab      AF Lab ≡ minimal (conflictfreeLab AF) Lab undec
```

The right-hand sides of `base.thy` and `simplified/labellings-simpl.thy`, without the type declarations.

Backup: the Nitpick settings that matter

Option	Effect in the session
`satisfy`	Search for a model of the statement, not for a countermodel
`card Arg=4, card Label=3`	Fix the sizes of the finite types to their exact values, so that the search scope is neither too small nor needlessly large
`box=false`	Switch off the boxing of function types. Without it Nitpick does not find the models of the definitions that nest `minimal` or `maximal`
`sat_solver=SAT4J`	The SAT back end; the only one available on Windows
`timeout=300`	For the ideal-extension queries, which take about a minute each with SAT4J
`expect=genuine`	Makes the build fail unless a genuine model or countermodel is found

- Every query has the same shape: a statement with a free variable, `nitpick` with `satisfy`, and `oops`

Backup: building and reading the session

Building

```
isabelle build -D .
```

- Run in the folder AA_Fixed, with Isabelle2025-2
- All 20 theories are checked, in about nine minutes on a laptop; skipped proofs are not accepted

Reading

- A statement closed with by or qed is a theorem
- A statement closed with oops is discarded: a model-generation query, a refuted converse, or an open question
- nitpick before oops: a model or a countermodel was found

Part of the talk	Theories
Building blocks, semantics	misc, base, extensions, labellings, and their simplified/ versions
Meta-theory	ext-properties, lab-properties, the two relationships theories, correspondence, adequacy, Zorn-lemma
Generation	simplified/model-generation