Friedrich-Alexander-Universität
Erlangen-Nürnberg

Inquisitive First-Order Logic, Bounded and Mechanized

Implementation of (Bounded) Inquisitive First-Order Logic in Rocq

Max Ole Elliger

June 17, 2025

Intuition

Inquisitive FOL can be seen as an extension of classical logic by *questions*.

Intuition

Inquisitive FOL can be seen as an extension of classical logic by *questions*.

Example

Natural Language	**Formula**
Luisa is guilty.	$\mathrm{Guilty}\,(\mathrm{Luisa})$
If Luisa was there, do we know whether Luisa is guilty?	$\mathrm{WasThere}\,(\mathrm{Luisa}) \rightarrow ?\,\mathrm{Guilty}\,(\mathrm{Luisa})$
If we knew whether Luisa was there, do we know whether Luisa is guilty?	$?\,\mathrm{WasThere}\,(\mathrm{Luisa}) \rightarrow ?\,\mathrm{Guilty}\,(\mathrm{Luisa})$
Is there some person, who is guilty?	$\exists\, x.\ \mathrm{Guilty}\,(x)$

Intuition

Formulae shall be *supported* by sets of possible worlds which refer to FO-Models.

Intuition

Formulae shall be *supported* by sets of possible worlds which refer to FO-Models.

Example

- Consider the following possible worlds regarding Luisa:

	Guilty	**Not Guilty**
Was There	w_1	w_2
Was Not There	w_3	w_4

Intuition

Formulae shall be *supported* by sets of possible worlds which refer to FO-Models.

Example

- Consider the following possible worlds regarding Luisa:

	Guilty	**Not Guilty**
Was There	w_1	w_2
Was Not There	w_3	w_4

- We get the following properties regarding the single worlds:

$$w_1 \models \mathrm{WasThere}\,(\mathrm{Luisa}) \rightarrow ?\,\mathrm{Guilty}\,(\mathrm{Luisa})$$
$$w_2 \models \mathrm{WasThere}\,(\mathrm{Luisa}) \rightarrow ?\,\mathrm{Guilty}\,(\mathrm{Luisa})$$
$$w_3 \models \mathrm{WasThere}\,(\mathrm{Luisa}) \rightarrow ?\,\mathrm{Guilty}\,(\mathrm{Luisa})$$
$$w_4 \models \mathrm{WasThere}\,(\mathrm{Luisa}) \rightarrow ?\,\mathrm{Guilty}\,(\mathrm{Luisa})$$

Intuition

Formulae shall be *supported* by sets of possible worlds which refer to FO-Models.

Example

- Consider the following possible worlds regarding Luisa:

	Guilty	Not Guilty
Was There	w_1	w_2
Was Not There	w_3	w_4

- We get the following properties regarding the single worlds:

$$w_1 \models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$
$$w_2 \models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$
$$w_3 \models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$
$$w_4 \models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$

- If we look at *information states*, we get the following support properties:

$$\{w_1, w_2\} \not\models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$
$$\{w_1, w_3\} \models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$
$$\{w_1, w_2, w_3\} \not\models \text{WasThere}\,(\text{Luisa}) \rightarrow ?\,\text{Guilty}\,(\text{Luisa})$$

Syntax [1]

Definition

- We call a set $\Sigma := (\mathsf{P}_\Sigma, \mathsf{F}_\Sigma, \mathrm{ar}_\Sigma, \mathrm{rigid}_\Sigma)$ a *signature*.
- P_Σ provides *predicate symbols*.
- F_Σ provides *function symbols*.
- $\mathrm{ar}_\Sigma \colon \mathsf{P}_\Sigma + \mathsf{F}_\Sigma \to \mathbb{N}$ maps symbols to their *arity*.
- $\mathrm{rigid}_\Sigma \subseteq \mathsf{F}_\Sigma$ indicates whether a function symbol is *rigid*.

[1] [Cia22]

Syntax [1]

Definition

- We call a set $\Sigma := (\mathsf{P}_\Sigma, \mathsf{F}_\Sigma, \mathrm{ar}_\Sigma, \mathrm{rigid}_\Sigma)$ a *signature*.
- P_Σ provides *predicate symbols*.
- F_Σ provides *function symbols*.
- $\mathrm{ar}_\Sigma \colon \mathsf{P}_\Sigma + \mathsf{F}_\Sigma \to \mathbb{N}$ maps symbols to their *arity*.
- $\mathrm{rigid}_\Sigma \subseteq \mathsf{F}_\Sigma$ indicates whether a function symbol is *rigid*.

Assume the existence of a set Var of *variables*.

[1] [Cia22]

Syntax [1]

Definition

- We call a set $\Sigma := (\mathsf{P}_\Sigma, \mathsf{F}_\Sigma, \mathrm{ar}_\Sigma, \mathrm{rigid}_\Sigma)$ a *signature*.
- P_Σ provides *predicate symbols*.
- F_Σ provides *function symbols*.
- $\mathrm{ar}_\Sigma \colon \mathsf{P}_\Sigma + \mathsf{F}_\Sigma \to \mathbb{N}$ maps symbols to their *arity*.
- $\mathrm{rigid}_\Sigma \subseteq \mathsf{F}_\Sigma$ indicates whether a function symbol is *rigid*.

Assume the existence of a set Var of *variables*.

Definition

Terms and *Formulae* over a signature Σ are defined as follows:

$$t \in \mathrm{Ter}_\Sigma ::= x \mid f\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(f)}\right) \qquad f \in \mathsf{F}_\Sigma$$

$$\phi, \psi \in \mathcal{F}_\Sigma ::= P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) \mid \bot \mid \phi \to \psi \mid \phi \wedge \psi \mid \phi \mathbin{\rotatebox[origin=c]{0}{$\vee\!\!\!\vee$}} \psi \mid \forall x.\, \phi \mid \exists x.\, \phi \qquad P \in \mathsf{P}_\Sigma$$

$$?\phi := \phi \mathbin{\vee\!\!\!\vee} \neg\phi$$

[1] [Cia22]

Syntax
Regarding Rocq

[1] [dBr72]
[2] [STS]

Syntax
Regarding Rocq

- Implement variables via *De Bruijn indices*[1]:

$$\mathrm{Var} := \mathbb{N}$$
$$\phi \in \mathcal{F}_\Sigma ::= \ \ldots \mid \forall.\, \phi \mid \exists\,.\, \phi$$

[1] [dBr72]
[2] [STS]

Syntax
Regarding Rocq

- Implement variables via *De Bruijn indices*[1]:

$$\mathrm{Var} := \mathbb{N}$$
$$\phi \in \mathcal{F}_\Sigma ::= \ \dots \mid \forall .\, \phi \mid \exists \,.\, \phi$$

- Use Autosubst[2]-library to implement substitutions.

[1] [dBr72]
[2] [STS]

Syntax
Regarding Rocq

- Implement variables via *De Bruijn indices*[1]:

$$\mathrm{Var} := \mathbb{N}$$
$$\phi \in \mathcal{F}_\Sigma ::= \ \dots \mid \forall.\, \phi \mid \exists\,.\, \phi$$

- Use Autosubst[2]-library to implement substitutions.
- Use *Typeclasses* to use signatures implicitly.

[1] [dBr72]
[2] [STS]

Syntax
Regarding Rocq

- Implement variables via *De Bruijn indices*[1]:

$$\mathrm{Var} := \mathbb{N}$$
$$\phi \in \mathcal{F}_\Sigma ::= \ \ldots \mid \forall.\, \phi \mid \exists\, .\, \phi$$

- Use Autosubst[2]-library to implement substitutions.
- Use *Typeclasses* to use signatures implicitly.

- Ensure *decidable equality* for predicate and function symbols to distinguish them.

[1] [dBr72]
[2] [STS]

Syntax
Regarding Rocq

- Ensure *decidable equality* for predicate and function symbols to distinguish them.
- Implement arities via *arity types*:

$$\mathrm{ar}_\Sigma\colon \mathsf{P}_\Sigma + \mathsf{F}_\Sigma \to |\mathbf{FinSet}|$$

- Implement variables via *De Bruijn indices*[1]:

$$\mathrm{Var} := \mathbb{N}$$
$$\phi \in \mathcal{F}_\Sigma ::= \ \ldots \mid \forall.\,\phi \mid \exists\,.\,\phi$$

- Use Autosubst[2]-library to implement substitutions.
- Use *Typeclasses* to use signatures implicitly.

[1] [dBr72]
[2] [STS]

Syntax
Regarding Rocq

- Implement variables via *De Bruijn indices*[1]:

$$\mathrm{Var} := \mathbb{N}$$
$$\phi \in \mathcal{F}_\Sigma ::= \ \ldots \mid \forall .\, \phi \mid \exists \,.\, \phi$$

- Use Autosubst[2]-library to implement substitutions.
- Use *Typeclasses* to use signatures implicitly.

- Ensure *decidable equality* for predicate and function symbols to distinguish them.
- Implement arities via *arity types*:

$$\mathrm{ar}_\Sigma \colon \mathsf{P}_\Sigma + \mathsf{F}_\Sigma \to |\mathbf{FinSet}|$$

- Implement arguments via *argument functions*:

$$t ::= \ldots \mid f\,(args) \quad \text{where } args \ : \mathrm{ar}_\Sigma\,(f) \quad \to \mathrm{Ter}_\Sigma$$
$$\phi ::= P\,(args) \mid \ldots \quad \text{where } args \ : \mathrm{ar}_\Sigma\,(P) \quad \to \mathrm{Ter}_\Sigma$$

1 [dBr72]
2 [STS]

Syntax

Regarding Rocq (2)

```
Class Signature :=
  {
    PSymb : Type;
    PSymb_EqDec :: EqDec (eq_setoid PSymb);
    PAri : PSymb → Type;
    FSymb : Type;
    FSymb_EqDec :: EqDec (eq_setoid FSymb);
    FAri : FSymb → Type;
    rigid : FSymb → bool
    (* ... *)
  }.

Inductive form `{Signature} :=
  | Pred : forall (p : PSymb), (PAri p → term) → form
  | Bot : var → form
  | Impl : form → form → form
  | Conj : form → form → form
  | Idisj : form → form → form
  | Forall : {bind term in form} → form
  | Iexists : {bind term in form} → form.
```

Syntax

Regarding Rocq (2)

```
Class Signature :=
  {
    PSymb : Type;
    PSymb_EqDec :: EqDec (eq_setoid PSymb);
    PAri : PSymb → Type;
    FSymb : Type;
    FSymb_EqDec :: EqDec (eq_setoid FSymb);
    FAri : FSymb → Type;
    rigid : FSymb → bool
    (* ... *)
  }.

Inductive form ‘{Signature} :=
  | Pred : forall (p : PSymb), (PAri p → term) → form
  | Bot : var → form
  | Impl : form → form → form
  | Conj : form → form → form
  | Idisj : form → form → form
  | Forall : {bind term in form} → form
  | Iexists : {bind term in form} → form.
```

- (Decidable) syntactic equality for formulae (and terms) becomes non-trivial because of dependent types.

Syntax

Regarding Rocq (2)

```
Class Signature :=
  {
    PSymb : Type;
    PSymb_EqDec :: EqDec (eq_setoid PSymb);
    PAri : PSymb → Type;
    FSymb : Type;
    FSymb_EqDec :: EqDec (eq_setoid FSymb);
    FAri : FSymb → Type;
    rigid : FSymb → bool
    (* ... *)
  }.

Inductive form `{Signature} :=
  | Pred : forall (p : PSymb), (PAri p → term) → form
  | Bot : var → form
  | Impl : form → form → form
  | Conj : form → form → form
  | Idisj : form → form → form
  | Forall : {bind term in form} → form
  | Iexists : {bind term in form} → form.
```

- (Decidable) syntactic equality for formulae (and terms) becomes non-trivial because of dependent types.
- Solution: Define a setoid equality for terms and formulae.

Syntax

Regarding Rocq (3)

```
Fixpoint term_eq `{S : Signature} (t : term) : term → Prop :=
  match t with
  | Var x1 ⇒
      fun t2 ⇒
      match t2 with
      | Var x2 ⇒ (x1 == x2)%type
      | _ ⇒ False
      end
  | Func f1 args1 ⇒
      fun t2 ⇒
      match t2 with
      | Func f2 args2 ⇒
          match equiv_dec f1 f2 with
          | left Heq ⇒
              term_eq_Func_Func_EqDec term_eq f1 args1 f2 args2 Heq
          | _ ⇒ False
          end
      | _ ⇒ False
      end
  end.
```

```
Definition term_eq_Func_Func_EqDec
  `{ S : Signature}
  (rec : relation term)
  (f1 : FSymb)
  (args1 : FAri f1 → term)
  (f2 : FSymb)
  (args2 : FAri f2 → term)
  (is_equal : (f1 == f2)%type) : Prop :=

  eq_rect
  f1
  (fun f ⇒ (FAri f → term) → Prop)
  (fun args ⇒
    forall arg,
      rec (args1 arg) (args arg)
  )
  f2
  is_equal
  args2.
```

Semantics [1]
Models, States

Definition

Let Σ be a signature.

- A tuple $\mathfrak{M} := \left(\mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, (\mathfrak{M}_w \llbracket f \rrbracket)_{w \in W, f \in \mathsf{F}_\Sigma}, (\mathfrak{M}_w \llbracket P \rrbracket)_{w \in W, P \in \mathsf{P}_\Sigma}\right)$ is called a *model*.

[1] [Cia22]

Semantics [1]

Models, States

Definition

Let Σ be a signature.

- A tuple $\mathfrak{M} := \left(\mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, (\mathfrak{M}_w [\![f]\!])_{w \in W, f \in \mathsf{F}_\Sigma}, (\mathfrak{M}_w [\![P]\!])_{w \in W, P \in \mathsf{P}_\Sigma}\right)$ is called a *model*.
- $\mathrm{W}_{\mathfrak{M}}$ is a set of *possible worlds*.

[1] [Cia22]

Semantics [1]

Models, States

Definition

Let Σ be a signature.

- A tuple $\mathfrak{M} := \left(\mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, (\mathfrak{M}_w \llbracket f \rrbracket)_{w \in W, f \in \mathsf{F}_\Sigma}, (\mathfrak{M}_w \llbracket P \rrbracket)_{w \in W, P \in \mathsf{P}_\Sigma}\right)$ is called a *model*.
- $\mathrm{W}_{\mathfrak{M}}$ is a set of *possible worlds*.
- $\mathrm{I}_{\mathfrak{M}}$ is a (non-empty) set of *individuals*.
- $\mathfrak{M}_w \llbracket f \rrbracket : \mathrm{I}_{\mathfrak{M}}^{\mathrm{ar}_\Sigma(f)} \to \mathrm{I}_{\mathfrak{M}}$ is the interpretation of f in a world w.
- $\mathfrak{M}_w \llbracket P \rrbracket \subseteq \mathrm{I}_{\mathfrak{M}}^{\mathrm{ar}_\Sigma(P)}$ is the interpretation of P in a world w.

[1] [Cia22]

Semantics [1]

Models, States

Definition

Let Σ be a signature.

- A tuple $\mathfrak{M} := \left(\mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, (\mathfrak{M}_w \llbracket f \rrbracket)_{w \in W, f \in \mathsf{F}_\Sigma}, (\mathfrak{M}_w \llbracket P \rrbracket)_{w \in W, P \in \mathsf{P}_\Sigma}\right)$ is called a *model*.
- $\mathrm{W}_{\mathfrak{M}}$ is a set of *possible worlds*.
- $\mathrm{I}_{\mathfrak{M}}$ is a (non-empty) set of *individuals*.
- $\mathfrak{M}_w \llbracket f \rrbracket : \mathrm{I}_{\mathfrak{M}}^{\mathrm{ar}_\Sigma(f)} \to \mathrm{I}_{\mathfrak{M}}$ is the interpretation of f in a world w.
- $\mathfrak{M}_w \llbracket P \rrbracket \subseteq \mathrm{I}_{\mathfrak{M}}^{\mathrm{ar}_\Sigma(P)}$ is the interpretation of P in a world w.
- for every rigid $f \in \mathsf{F}_\Sigma$ and for all $w_1, w_2 \in \mathrm{W}_{\mathfrak{M}}$ we have $\mathfrak{M}_{w_1} \llbracket f \rrbracket = \mathfrak{M}_{w_2} \llbracket f \rrbracket$.

[1] [Cia22]

Semantics [1]

Models, States

Definition

Let Σ be a signature.

- A tuple $\mathfrak{M} := \left(\mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, (\mathfrak{M}_w \llbracket f \rrbracket)_{w \in W, f \in \mathsf{F}_\Sigma}, (\mathfrak{M}_w \llbracket P \rrbracket)_{w \in W, P \in \mathsf{P}_\Sigma}\right)$ is called a *model*.
- $\mathrm{W}_{\mathfrak{M}}$ is a set of *possible worlds*.
- $\mathrm{I}_{\mathfrak{M}}$ is a (non-empty) set of *individuals*.
- $\mathfrak{M}_w \llbracket f \rrbracket : \mathrm{I}_{\mathfrak{M}}^{\mathrm{ar}_\Sigma(f)} \to \mathrm{I}_{\mathfrak{M}}$ is the interpretation of f in a world w.
- $\mathfrak{M}_w \llbracket P \rrbracket \subseteq \mathrm{I}_{\mathfrak{M}}^{\mathrm{ar}_\Sigma(P)}$ is the interpretation of P in a world w.
- for every rigid $f \in \mathsf{F}_\Sigma$ and for all $w_1, w_2 \in \mathrm{W}_{\mathfrak{M}}$ we have $\mathfrak{M}_{w_1} \llbracket f \rrbracket = \mathfrak{M}_{w_2} \llbracket f \rrbracket$.

Definition

Let Σ be a signature, $\mathfrak{M}$ be a model. A subset $s \subseteq \mathrm{W}_{\mathfrak{M}}$ is called an *(information) state*.

[1] [Cia22]

Semantics
Referent of a Term

Definition

Let Σ be a signature, $\mathfrak{M}$ be a Model, $s \subseteq \mathrm{W}_{\mathfrak{M}}$ an information state and $\eta\colon\ \mathrm{Var} \to \mathrm{I}_{\mathfrak{M}}$ a variable assignment. The *referent* of a term $t \in \mathrm{Ter}_{\Sigma}$ is defined as follows:

$$\mathfrak{M}_{w,\eta}\left[\!\left[x\right]\!\right] := \eta\left(x\right)$$

$$\mathfrak{M}_{w,\eta}\left[\!\left[f\left(t_1,\ldots,t_{\mathrm{ar}_{\Sigma}(f)}\right)\right]\!\right] := \mathfrak{M}_{w}\left[\!\left[f\right]\!\right]\left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right],\ldots,\mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_{\Sigma}(f)}\right]\!\right]\right)$$

Semantics
Referent of a Term

Definition

Let Σ be a signature, $\mathfrak{M}$ be a Model, $s \subseteq \mathrm{W}_{\mathfrak{M}}$ an information state and $\eta\colon\ \mathrm{Var} \to \mathrm{I}_{\mathfrak{M}}$ a variable assignment. The *referent* of a term $t \in \mathrm{Ter}_{\Sigma}$ is defined as follows:

$$\mathfrak{M}_{w,\eta}\left[\!\left[x\right]\!\right] := \eta\left(x\right)$$
$$\mathfrak{M}_{w,\eta}\left[\!\left[f\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(f)}\right)\right]\!\right] := \mathfrak{M}_{w}\left[\!\left[f\right]\!\right]\left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(f)}\right]\!\right]\right)$$

Using the new syntax:

$$\mathfrak{M}_{w,\eta}\left[\!\left[f\left(args\right)\right]\!\right] := \mathfrak{M}_{w}\left[\!\left[f\right]\!\right]\left(\mathfrak{M}_{w,\eta}\left[\!\left[-\right]\!\right] \circ args\right)$$

Semantics
Support

Definition

The *support* relation $\models$ is defined as follows:

Semantics
Support

Definition

The *support* relation $\models$ is defined as follows:

$$\mathfrak{M}, s, \eta \models P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) :\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(P)}\right]\!\right]\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right]$$

Semantics
Support

Definition
The *support* relation $\models$ is defined as follows:

$$\mathfrak{M}, s, \eta \models P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) :\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(P)}\right]\!\right]\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right]$$

$$\mathfrak{M}, s, \eta \models \bot :\Longleftrightarrow s = \emptyset$$

Semantics
Support

Definition

The *support* relation $\models$ is defined as follows:

$$\begin{aligned}
\mathfrak{M}, s, \eta \models P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) &:\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(P)}\right]\!\right]\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right] \\
\mathfrak{M}, s, \eta \models \bot &:\Longleftrightarrow s = \emptyset \\
\mathfrak{M}, s, \eta \models \phi \to \psi &:\Longleftrightarrow \text{for all } t \subseteq s,\ \mathfrak{M}, t, \eta \models \phi \text{ implies } \mathfrak{M}, t, \eta \models \psi
\end{aligned}$$

Semantics
Support

Definition

The *support* relation $\models$ is defined as follows:

$$\begin{aligned}
\mathfrak{M}, s, \eta \models P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) &:\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(P)}\right]\!\right]\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right] \\
\mathfrak{M}, s, \eta \models \bot &:\Longleftrightarrow s = \emptyset \\
\mathfrak{M}, s, \eta \models \phi \to \psi &:\Longleftrightarrow \text{for all } t \subseteq s,\ \mathfrak{M}, t, \eta \models \phi \text{ implies } \mathfrak{M}, t, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \phi \wedge \psi &:\Longleftrightarrow \mathfrak{M}, s, \eta \models \phi \text{ and } \mathfrak{M}, s, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \phi \mathbin{\rotatebox[origin=c]{180}{$\bar{\wedge}$}} \psi &:\Longleftrightarrow \mathfrak{M}, s, \eta \models \phi \text{ or } \mathfrak{M}, s, \eta \models \psi
\end{aligned}$$

Semantics
Support

Definition

The *support* relation $\models$ is defined as follows:

$$
\begin{aligned}
\mathfrak{M}, s, \eta \models P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) &:\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(P)}\right]\!\right]\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right] \\
\mathfrak{M}, s, \eta \models \bot &:\Longleftrightarrow s = \emptyset \\
\mathfrak{M}, s, \eta \models \phi \to \psi &:\Longleftrightarrow \text{for all } t \subseteq s,\ \mathfrak{M}, t, \eta \models \phi \text{ implies } \mathfrak{M}, t, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \phi \wedge \psi &:\Longleftrightarrow \mathfrak{M}, s, \eta \models \phi \text{ and } \mathfrak{M}, s, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \phi \vee\!\!\!\vee \psi &:\Longleftrightarrow \mathfrak{M}, s, \eta \models \phi \text{ or } \mathfrak{M}, s, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \forall x.\, \phi &:\Longleftrightarrow \text{for all } i \in \mathrm{I}_{\mathfrak{M}},\ \mathfrak{M}, s, \eta\left[x \mapsto i\right] \models \phi \\
\mathfrak{M}, s, \eta \models \exists x.\, \phi &:\Longleftrightarrow \text{there exists } i \in \mathrm{I}_{\mathfrak{M}},\ \mathfrak{M}, s, \eta\left[x \mapsto i\right] \models \phi
\end{aligned}
$$

Semantics
Support

Definition

The *support* relation $\models$ is defined as follows:

$$
\begin{aligned}
\mathfrak{M}, s, \eta \models P\left(t_1, \ldots, t_{\mathrm{ar}_\Sigma(P)}\right) &:\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[t_1\right]\!\right], \ldots, \mathfrak{M}_{w,\eta}\left[\!\left[t_{\mathrm{ar}_\Sigma(P)}\right]\!\right]\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right] \\
\mathfrak{M}, s, \eta \models \bot &:\Longleftrightarrow s = \emptyset \\
\mathfrak{M}, s, \eta \models \phi \to \psi &:\Longleftrightarrow \text{for all } t \subseteq s,\ \mathfrak{M}, t, \eta \models \phi \text{ implies } \mathfrak{M}, t, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \phi \wedge \psi &:\Longleftrightarrow \mathfrak{M}, s, \eta \models \phi \text{ and } \mathfrak{M}, s, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \phi \mathbin{\rlap{\vee}\,\vee} \psi &:\Longleftrightarrow \mathfrak{M}, s, \eta \models \phi \text{ or } \mathfrak{M}, s, \eta \models \psi \\
\mathfrak{M}, s, \eta \models \forall x.\, \phi &:\Longleftrightarrow \text{for all } i \in \mathrm{I}_{\mathfrak{M}},\ \mathfrak{M}, s, \eta\left[x \mapsto i\right] \models \phi \\
\mathfrak{M}, s, \eta \models \exists x.\, \phi &:\Longleftrightarrow \text{there exists } i \in \mathrm{I}_{\mathfrak{M}},\ \mathfrak{M}, s, \eta\left[x \mapsto i\right] \models \phi
\end{aligned}
$$

Using the new syntax:

$$
\begin{aligned}
\mathfrak{M}, s, \eta \models P\left(args\right) &:\Longleftrightarrow \text{for all } w \in s \text{ we have } \left(\mathfrak{M}_{w,\eta}\left[\!\left[-\right]\!\right] \circ args\right) \in \mathfrak{M}_w\left[\!\left[P\right]\!\right] \\
\mathfrak{M}, s, \eta \models \forall.\, \phi &:\Longleftrightarrow \text{for all } i \in \mathrm{I}_{\mathfrak{M}},\ \mathfrak{M}, s, i \bullet \eta \models \phi \\
\mathfrak{M}, s, \eta \models \exists.\, \phi &:\Longleftrightarrow \text{there exists } i \in \mathrm{I}_{\mathfrak{M}},\ \mathfrak{M}, s, i \bullet \eta \models \phi
\end{aligned}
$$

Semantics

Various properties

Persistency

$t \subseteq s$ and $\mathfrak{M}, s, \eta \models \phi \Longrightarrow \mathfrak{M}, t, \eta, \models \phi$

Empty State Property

$$\mathfrak{M}, \emptyset, \eta \models \phi$$

Semantics

Various properties

Persistency

$t \subseteq s$ and $\mathfrak{M}, s, \eta \models \phi \Longrightarrow \mathfrak{M}, t, \eta, \models \phi$

Empty State Property

$$\mathfrak{M}, \emptyset, \eta \models \phi$$

- $\mathfrak{M}|_s := (s \subseteq \mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, \ldots)$

Locality

$$\mathfrak{M}, s, \eta \models \phi \Longleftrightarrow \mathfrak{M}|_s, s, \eta \models \phi$$

Semantics

Various properties

Persistency

$t \subseteq s$ and $\mathfrak{M}, s, \eta \models \phi \implies \mathfrak{M}, t, \eta, \models \phi$

Empty State Property

$$\mathfrak{M}, \emptyset, \eta \models \phi$$

- $\mathfrak{M}|_s := (s \subseteq \mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, \ldots)$

Locality

$$\mathfrak{M}, s, \eta \models \phi \iff \mathfrak{M}|_s, s, \eta \models \phi$$

- Defining $\mathfrak{M}|_s$ in Rocq needs subtypes.
- Solution: Generalize $\mathrm{W}_{\mathfrak{M}}$ to a *setoid*.

Semantics

Various properties

Persistency

$$t \subseteq s \text{ and } \mathfrak{M}, s, \eta \models \phi \Longrightarrow \mathfrak{M}, t, \eta, \models \phi$$

Empty State Property

$$\mathfrak{M}, \emptyset, \eta \models \phi$$

- $\mathfrak{M}|_s := (s \subseteq \mathrm{W}_{\mathfrak{M}}, \mathrm{I}_{\mathfrak{M}}, \ldots)$

Locality

$$\mathfrak{M}, s, \eta \models \phi \Longleftrightarrow \mathfrak{M}|_s, s, \eta \models \phi$$

- Defining $\mathfrak{M}|_s$ in Rocq needs subtypes.
- Solution: Generalize $\mathrm{W}_{\mathfrak{M}}$ to a *setoid*.

```
Context `{M : Model}. Context (s : state).

Program Definition restricted_Model : Model :=
  {|
    World := {w : World | contains s w};
    World_Setoid := sig_Setoid (contains_Morph s);
    PInterpretation w := PInterpretation (proj1_sig w);
    FInterpretation w := FInterpretation (proj1_sig w);
    (* ... *)
 |}.

Program Definition restricted_state (t : state) :
  @state _ (restricted_Model s) := (* ... *)

Program Definition unrestricted_state
  (t : @state _ (restricted_Model s)) : state := (* ... *)

Proposition locality `{M : Model} :
  forall phi s a t, substate t s →
    support phi t a ↔ support phi (@restricted_state _ M s t) a.
```

Semantics
InqFOL

Definition

Define *Inquisitive First-Order Logic* as follows:

$$\textbf{InqLog}_{\Sigma} := \{\phi \in \mathcal{F}_{\Sigma} \mid \mathfrak{M}, s, \eta \models \phi \text{ for all models } \mathfrak{M}, s \subseteq \mathrm{W}_{\mathfrak{M}}, \eta\colon\ \mathrm{Var} \to \mathrm{I}_{\mathfrak{M}}\}$$

[1] [CG22]

Semantics
InqFOL

Definition

Define *Inquisitive First-Order Logic* as follows:

$$\mathbf{InqLog}_\Sigma := \{\phi \in \mathcal{F}_\Sigma \mid \mathfrak{M}, s, \eta \models \phi \text{ for all models } \mathfrak{M}, s \subseteq \mathrm{W}_\mathfrak{M}, \eta\colon \mathrm{Var} \to \mathrm{I}_\mathfrak{M}\}$$

- There exists a ND-System by Ciardelli/Grilletti[1] which is sound, but not yet proven to be complete.

[1] [CG22]

Boundedness
Introduction

- Restricting the set of worlds to be finite yields *Bounded Inquisitive FOL*.

$$\mathbf{InqLogB}_{\Sigma,n} := \{\phi \in \mathcal{F}_\Sigma \mid \mathfrak{M}, s, \eta \models \phi \text{ for all models } \mathfrak{M} \text{ with } |\mathrm{W}_\mathfrak{M}| < n, s \subseteq \mathrm{W}_\mathfrak{M}, \eta\colon \mathrm{Var} \to \mathrm{I}_\mathfrak{M}\}$$

$$\mathbf{InqLogB}_{\Sigma} := \bigcap_{n\in\mathbb{N}} \mathbf{InqLogB}_{\Sigma,n}$$

$$= \{\phi \in \mathcal{F}_\Sigma \mid \mathfrak{M}, s, \eta \models \phi \text{ for all models } \mathfrak{M}, s \subseteq_{\text{fin}} \mathrm{W}_\mathfrak{M}, \eta\colon \mathrm{Var} \to \mathrm{I}_\mathfrak{M}\}$$

[1] [CG22]

Boundedness
Introduction

- Restricting the set of worlds to be finite yields *Bounded Inquisitive FOL*.

$$\mathbf{InqLogB}_{\Sigma,n} := \{\phi \in \mathcal{F}_\Sigma \mid \mathfrak{M}, s, \eta \models \phi \text{ for all models } \mathfrak{M} \text{ with } |\mathrm{W}_\mathfrak{M}| < n, s \subseteq \mathrm{W}_\mathfrak{M}, \eta\colon \mathrm{Var} \to \mathrm{I}_\mathfrak{M}\}$$

$$\mathbf{InqLogB}_{\Sigma} := \bigcap_{n\in\mathbb{N}} \mathbf{InqLogB}_{\Sigma,n}$$

$$= \{\phi \in \mathcal{F}_\Sigma \mid \mathfrak{M}, s, \eta \models \phi \text{ for all models } \mathfrak{M}, s \subseteq_{\text{fin}} \mathrm{W}_\mathfrak{M}, \eta\colon \mathrm{Var} \to \mathrm{I}_\mathfrak{M}\}$$

- Ciardelli/Griletti[1] extended their ND-System for $\mathbf{InqLogB}_{\Sigma,n}$ and it proved it to be also complete *(for most signatures)*.
- Added axiom: *Cardinality Formula*, which depends on the concrete signature.

[1] [CG22]

Cardinality Formulae[1]

Only One Predicate

$$C_0^{\{P\}} := \bot$$

$$C_1^{\{P\}} := \forall x ?Px$$

$$C_{n+1}^{\{P\}} := \exists x \bigvee_{i=1}^{n} \left[(Px \rightarrow C_i^{\{P\}}) \wedge (\neg Px \rightarrow C_{n+1-i}^{\{P\}}) \right]$$

[1] [CG22]

Cardinality Formulae[1]

Assuming all function symbols are rigid

$$
\begin{aligned}
C_0^\Sigma &:= \bot \\
C_1^\Sigma &:= \forall \overline{x}_1 ?R_1(\overline{x}_1) \land \ldots \land \forall \overline{x}_l ?R_l(\overline{x}_l) \\
C_{n+1}^\Sigma &:= \exists \overline{x}_1 \bigvee\!\!\!\!\bigvee_{i=1}^{n} \left[(R_1(\overline{x}_1) \to C_i^\Sigma) \land (\neg R_1(\overline{x}_1) \to C_{n+1-i}^\Sigma) \right] \;\mathbin{\rotatebox[origin=c]{180}{$\bigwedge$}}\!\!\!\vee \ldots \\
&\quad \ldots \vee\!\!\!\vee \exists \overline{x}_l \bigvee\!\!\!\!\bigvee_{i=1}^{n} \left[(R_l(\overline{x}_l) \to C_i^\Sigma) \land (\neg R_l(\overline{x}_l) \to C_{n+1-i}^\Sigma) \right]
\end{aligned}
$$

[1] [CG22]

Cardinality Formulae[1]

Adding equality to the syntax

$$C_0^\Sigma := \bot$$

$$C_1^\Sigma := \bigwedge_{j=1}^{l} \forall \overline{x}_j ?R_j(\overline{x}_j) \;\wedge\; \bigwedge_{j=1}^{h} \forall \overline{y}_j \exists z (f_j(\overline{y}_j) = z)$$

$$C_{n+1}^\Sigma := \mathbb{V}_{j=1}^{l} \exists \overline{x}_j \mathbb{V}_{i=1}^{n} [\, (R_j(\overline{x}_j) \to C_i^\Sigma) \wedge (\neg R_j(\overline{x}_j) \to C_{n+1-i}^\Sigma) \,] \;\mathbb{V}$$

$$\mathbb{V}\; \mathbb{V}_{j=1}^{h} \exists \overline{y}_j z \mathbb{V}_{i=1}^{n} [\, (f_j(\overline{y}_j) = z \to C_i^\Sigma) \;\wedge\; (f_j(\overline{y}_j) \neq z \to C_{n+1-i}^\Sigma) \,]$$

[1] [CG22]

A Sequent Calculus

- Litak/Sano provide a Sequent Calculus[1] for $\mathbf{InqLogB}_{\Sigma}$ which is proven to be sound and complete.

[1] [LS]

A Sequent Calculus

- Litak/Sano provide a Sequent Calculus[1] for $\mathbf{InqLogB}_{\Sigma}$ which is proven to be sound and complete.
- Labels: Finite sets of natural numbers.

[1] [LS]

A Sequent Calculus

- Litak/Sano provide a Sequent Calculus[1] for $\mathbf{InqLogB}_{\Sigma}$ which is proven to be sound and complete.
- Labels: Finite sets of natural numbers.
- Sequents: $\Gamma \Rightarrow \Delta$ where
  - Γ, Δ are finite sets of labelled formulae, e.g. $(\{1, 2\}, \phi)$
  - Γ: “Assumptions”
  - Δ: “Possible Proof Goals”

[1] [LS]

A Sequent Calculus

- Litak/Sano provide a Sequent Calculus[1] for $\mathbf{InqLogB}_{\Sigma}$ which is proven to be sound and complete.
- Labels: Finite sets of natural numbers.
- Sequents: $\Gamma \Rightarrow \Delta$ where
  - Γ, Δ are finite sets of labelled formulae, e.g. $(\{1, 2\}, \phi)$
  - Γ: “Assumptions”
  - Δ: “Possible Proof Goals”
- Semantics of a labelled formula (X, ϕ) are given by a mapping $f : \mathbb{N} \to \mathrm{W}_{\mathfrak{M}}$.
- Semantics of a Sequent $\Gamma \Rightarrow \Delta$:

$$\begin{array}{rll} \text{If } \mathfrak{M}, f, \eta \models (X, \phi) & \text{for all} & (X, \phi) \in \Gamma, \\ \text{then } \mathfrak{M}, f, \eta \models (X, \psi) & \text{for some} & (Y, \phi) \in \Delta \end{array}$$

[1] [LS]

A Sequent Calculus

- Litak/Sano provide a Sequent Calculus[1] for $\mathbf{InqLogB}_{\Sigma}$ which is proven to be sound and complete.
- Labels: Finite sets of natural numbers.
- Sequents: $\Gamma \Rightarrow \Delta$ where
  - Γ, Δ are finite sets of labelled formulae, e.g. $(\{1, 2\}, \phi)$
  - Γ: "Assumptions"
  - Δ: "Possible Proof Goals"
- Semantics of a labelled formula (X, ϕ) are given by a mapping $f : \mathbb{N} \to \mathrm{W}_{\mathfrak{M}}$.
- Semantics of a Sequent $\Gamma \Rightarrow \Delta$:

$$\begin{array}{rll} \text{If } \mathfrak{M}, f, \eta \models (X, \phi) & \text{for all} & (X, \phi) \in \Gamma, \\ \text{then } \mathfrak{M}, f, \eta \models (X, \psi) & \text{for some} & (Y, \phi) \in \Delta \end{array}$$

- We slightly adapt the Sequent Calculus of Litak/Sano to our needs.

[1] [LS]

A Sequent Calculus
Some Rules

$$\frac{(\emptyset, \phi) \in \Delta}{\Gamma \Rightarrow \Delta}\text{(empty)}$$

$$\frac{(X, \bot) \in \Gamma \quad n \in X}{\Gamma \Rightarrow \Delta}(\bot \Rightarrow)$$

$$\frac{(X, \phi \to \psi) \in \Delta \quad \{\Gamma, (Y, \phi) \Rightarrow (Y, \psi), \Delta \mid Y \subseteq X\}}{\Gamma \Rightarrow \Delta}(\Rightarrow\to)$$

$$\frac{(X, \phi \vee\!\!\!\vee \psi) \in \Delta \quad \Gamma \Rightarrow (X, \phi), (X, \psi), \Delta}{\Gamma \Rightarrow \Delta}(\Rightarrow \vee\!\!\!\vee)$$

$$\frac{(X, \phi \vee\!\!\!\vee \psi) \in \Gamma \quad \Gamma, (X, \phi) \Rightarrow \Delta \quad \Gamma, (X, \psi) \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}(\vee\!\!\!\vee \Rightarrow)$$

$$\frac{(X, \exists\!\!\!\exists\,.\,\phi) \in \Delta \quad t \text{ is rigid} \quad \Gamma \Rightarrow (X, \phi.\,[t \bullet \mathrm{ids}]), \Delta}{\Gamma \Rightarrow \Delta}(\Rightarrow \exists\!\!\!\exists)$$

$$\frac{(X, \exists\!\!\!\exists\,.\,\phi) \in \Gamma \quad \Gamma.\,[(+1)], (X, \phi) \Rightarrow \Delta.\,[(+1)]}{\Gamma \Rightarrow \Delta}(\exists\!\!\!\exists \Rightarrow)$$

Sequent Calculus

Some Notes

- The rule of cut is proven to be admissible by Litak/Sano.
- Inside our formalization, we hardcoded it whithout showing admissibility.
- Our implementation of the sequent calculus also comes with a proof of soundness.
- We currently lack of a proof of completeness.

```
Inductive Seq `{Signature} : relation (list lb_form) :=
  (* ... *)
  | Seq_Iexists_r :
      forall ls rs ns phi t,
        InS (pair ns <{iexists phi}>) rs →
        term_rigid t →
        Seq ls ((pair ns phi.|[t/]) :: rs) →
        Seq ls rs.

Theorem soundness `{Signature} :
  forall Phi Psi, Seq Phi Psi →
    satisfaction_conseq Phi Psi.
Proof.
  induction 1. (* on Seq Phi Psi *)
  all: eauto using
    satisfaction_conseq_empty,
    satisfaction_conseq_id,
    (* ... *).
Qed.
```

Truth Semantics

- Define *Truth Semantics* via support of singleton states:

$$\mathfrak{M}, w, \eta \models_{\mathsf{truth}} \phi :\Longleftrightarrow \mathfrak{M}, \{w\}, \eta \models \phi$$

Truth Semantics

- Define *Truth Semantics* via support of singleton states:

$$\mathfrak{M}, w, \eta \models_{\mathsf{truth}} \phi :\Longleftrightarrow \mathfrak{M}, \{w\}, \eta \models \phi$$

- Truth semantics yield semantics of classic first-order logic.

Truth Semantics

- Define *Truth Semantics* via support of singleton states:

$$\mathfrak{M}, w, \eta \models_{\mathsf{truth}} \phi :\Longleftrightarrow \mathfrak{M}, \{w\}, \eta \models \phi$$

- Truth semantics yield semantics of classic first-order logic.
- Therefore, classic first-order logic is precisely $\mathbf{InqLogB}_{\Sigma,1}$.

Truth Semantics

- Define *Truth Semantics* via support of singleton states:

$$\mathfrak{M}, w, \eta \models_{\text{truth}} \phi :\Longleftrightarrow \mathfrak{M}, \{w\}, \eta \models \phi$$

- Truth semantics yield semantics of classic first-order logic.
- Therefore, classic first-order logic is precisely $\mathbf{InqLogB}_{\Sigma,1}$.

Example

$$\begin{aligned} \neg\neg P(0) \to P(0) & \quad \in \mathbf{InqLog}_{\Sigma} \\ \neg\neg\phi \to \phi & \quad \in \mathbf{InqLogB}_{\Sigma,1} \\ \neg\neg(P(0) \mathbin{\mathbb{V}} \neg P(0)) \to (P(0) \mathbin{\mathbb{V}} \neg P(0)) & \quad \notin \mathbf{InqLogB}_{\Sigma,2} \quad \supsetneq \mathbf{InqLogB}_{\Sigma} \end{aligned}$$

The Casari Scheme

- Consider the following so-called *Casari Scheme*:

$$\text{Casari} := (\forall.\ (\phi\,(0) \to \forall.\,\phi\,(0)) \to \forall.\,\phi\,(0)) \to \forall.\,\phi\,(0)$$

- We get the following properties:

$$(\forall.\ (P\,(0) \to \forall.\,P\,(0)) \to \forall.\,P\,(0)) \to \forall.\,P\,(0) \in \mathbf{InqLog}_{\Sigma}$$
$$(\forall.\ (\phi\,(0) \to \forall.\,\phi\,(0)) \to \forall.\,\phi\,(0)) \to \forall.\,\phi\,(0) \in \mathbf{InqLogB}_{\Sigma}$$
$$(\forall.\ ((\exists\,.\,R(1,0)) \to \forall.\ \exists\,.\,R(1,0)) \to \forall.\ \exists\,.\,R(1,0)) \to \forall.\ \exists\,.\,R(1,0) \notin \mathbf{InqLog}_{\Sigma}$$

The Casari Scheme
Regarding Schematic Bounded Validity

Theorem
The Casari Scheme is schematically bounded valid.[1]

Proof.
1. Prove that for every label X, the sequent $\Rightarrow (X, \mathrm{Casari})$ is derivable in the given sequent calculus.
2. By the rule $(\Rightarrow\rightarrow)$), it suffices to show for every $Y \subseteq X$ the derivability of the following sequent:

$$(Y, \forall.\ (\phi\,(0) \rightarrow \forall.\, \phi\,(0)) \rightarrow \forall.\, \phi\,(0)) \Rightarrow (Y, \forall.\, \phi\,(0))$$

3. Use wellfounded induction on Y to proceed. Proof uses the rule of cut.

□

[1] [LS]

The Casari Scheme

Regarding Schematic Validity

Theorem

The Casari Scheme is not schematically valid, e.g. Casari *instantiated with* $\phi := \exists\,.\,R(1,0)$ *is not schematically valid.*[1]

Proof Sketch.

By a suitable counterexample

[1] [LS]

The Casari Scheme
Regarding Schematic Validity

Theorem

The Casari Scheme is not schematically valid, e.g. Casari *instantiated with* $\phi := \exists\,.\,R(1,0)$ *is not schematically valid.*[1]

Proof Sketch.

By a suitable counterexample whose formalization just took 2 months . . . □

[1] [LS]

```
(** * The Casari "counter-example"

   We will now provide a counter-example to show that the
   Casari Scheme isn't schematically valid. For this, we
   need a concrete signature, a concret instance of the
   scheme via a formula [phi], a suitable model [M], a state
   [s] and a variable assignment [a] s.t. [M], [s] and [a]
   do not support [phi].
 *)
Module Casari_fails.

  Import PeanoNat.Nat.

  Local Arguments contains _ _ s w /.

  (** ** Signature and Syntax

     We will use our signature with a single binary
     predicate symbol for the counter example.
   *)
  Import Syntax_single_binary_predicate.

  (**
     The following formula will serve as our instance for
     the Casari Scheme:
   *)
  Definition IES : form :=
    <{iexists (Pred' (Var 1) (Var 0))}>.

  (**
     We can verify that [IES] has only one free variable.
   *)
  Remark highest_occ_free_var_IES :
    highest_occ_free_var IES (Some 0).
  Proof.
    intros sigma1 sigma2 H1.
    simpl.
    red.
    rewrite <- eq_rect_eq_dec; try exact PSymb_EqDec.
    intros [|]; try reflexivity.
    unfold mmap.
    unfold MMap_fun.
    unfold up.
    simpl.
```

```
  do 2 rewrite rename_subst'.
  rewrite H1; reflexivity.
Qed.

Print Assumptions highest_occ_free_var_IES.

(** ** The Model

   For our model, we decide on natural numbers to serve as
   our type of Worlds and Individuals. By this,
   [PInterpretation] becomes a ternary relation which we
   define before:
 *)
Definition rel (w m j : nat) : bool :=
  (
    negb (even m) &&
    (m =? j)
  ) ||
  (
    even m &&
    negb (j =? w) &&
    (
      negb (even j) ||
      (m <? j)
    )
  ).

(**
   We will now instantiate the model.
 *)

Local Obligation Tactic :=
  try decide equality;
  try contradiction.

Program Instance M : Model :=
  {|
    World := nat;
    World_Setoid := eq_setoid nat;
    Individual := nat;
    Individual_inh := 42;
    PInterpretation :=
      fun w p args =>
      rel w (args true) (args false)
  |}.

Next Obligation.
  intros w p args1 args2 H1.
```

```
  repeat rewrite H1.
  reflexivity.
Qed.

Next Obligation.
  intros w1 w2 H1.
  reflexivity.
Qed.

(** ** Intermezzo: Some classical logic properties *)

Lemma not_exists_forall_not {X} :
  forall (P : X -> Prop),
    ~ (exists x, P x) ->
    forall x,
      ~ P x.
Proof.
  firstorder.
Qed.

Lemma not_forall_exists_not {X} :
  forall (P : X -> Prop),
    ~ (forall x, P x) ->
    exists x,
      ~ P x.
Proof.
  intros P H1.
  apply NNPP.
  intros H2.
  apply H1.
  intros x.
  eapply not_exists_forall_not in H2.
  apply NNPP.
  exact H2.
Qed.

(** ** Some state properties

   We start by defining some notation for state
   properties.
 *)

Declare Custom Entry boolpred.

Notation "(? p ?)" := p
  (at level 0,
  p custom boolpred at level 99)
  : form_scope.
```

```
Notation "( x )" := x
  (in custom boolpred, x at level 99)
  : form_scope.

Notation "x" := x
  (in custom boolpred at level 0, x constr at level 0)
  : form_scope.

Notation "f x .. y" := (.. (f x) .. y)
  (in custom boolpred at level 0,
  only parsing,
  f constr at level 0,
  x constr at level 9,
  y constr at level 9)
  : form_scope.

Notation "p1 && p2" := (fun w => p1 w && p2 w)
  (in custom boolpred at level 40, right associativity)
  : form_scope.

Notation "p1 || p2" := (fun w => p1 w || p2 w)
  (in custom boolpred at level 50, right associativity)
  : form_scope.

Notation "~ p" := (fun w => negb (p w))
  (in custom boolpred at level 75)
  : form_scope.

(**
   We define [contains_all p] to say that s contains all
   worlds with property [p]. Note that this is in fact a
   duplicate of [substate] which is intended to
   distinguish between properties of worlds and states.
 *)

Definition contains_all (p : nat -> bool) (s : state) : Pr
  forall w,
    p w = true ->
    contains s w.

Instance contains_all_Proper :
  forall p,
    Proper (state_eq ==> iff) (contains_all p).
Proof.
  intros p s1 s2 H1.
  split.
  -
```

```
    intros H2 w H3.
    rewrite <- H1.
    apply H2.
    exact H3.
  -
    intros H2 w H3.
    rewrite H1.
    apply H2.
    exact H3.
Qed.

Lemma substate_contains_all :
  forall p s t,
    substate t s ->
    contains_all p t ->
    contains_all p s.
Proof.
  intros p s t H1 H2 w H3.
  apply H1.
  apply H2.
  exact H3.
Qed.

(**
   Next, we implement the notion that a state contains at
   least one world with property [p].
 *)
Definition contains_any (p : nat -> bool) (s : state) : Pr
  exists w,
    p w = true /\
    contains s w.

Instance contains_any_Proper :
  forall p,
    Proper (state_eq ==> iff) (contains_any p).
Proof.
  intros p s1 s2 H1.
  split.
  -
    intros [w [H2 H3]].
    exists w.
    rewrite <- H1.
    split; assumption.
  -
    intros [w [H2 H3]].
    exists w.
    rewrite H1.
    split; assumption.
```

```
(** ** Support for [IES]

   We start by analysing support for [IES] itself.
 *)

(**
   [support_IES_odd] represents Claim 3.7. in Litak/Sano
 *)
Proposition support_IES_odd :
  forall (s : state) (a : assignment),
    even (a 0) = false ->
    s, a |= IES.
Proof.
  intros s a H1.

  exists (a 0).

  intros w H2.
  simpl.
  unfold rel.

  rewrite H1.
  rewrite eqb_refl.
  reflexivity.
Qed.
```

```
    exact H6.
  -
    intros w H9.
    destruct (even w) eqn:HA.
    +
      specialize (H7 _ HA).
      destruct (contains_dec t w) as [HB|HB]; try assumptio
      rewrite contains_complement_iff in H7.
      specialize (H7 HB).
      apply leb_le in H7.
      apply ltb_lt in H9.
      lia.
    +
      apply H6.
      rewrite HA.
      reflexivity.
  -
    unfold CasariImpl in H3.
    rewrite support_Impl in H3.
    apply H3.
    +
      exact H5.
    +
      destruct (even (a 0)) eqn:H9.
      *
        apply support_IES_even.
        --
           exact H9.
        --
           destruct H8 as [e2 [H81 H82]].
           exists e2.
           simpl in *.
           rewrite H81,H82.
           rewrite orb_true_r.
           split; reflexivity.
      *
        apply support_IES_odd.
        exact H9.
Qed.

Print Assumptions support_CasariImpl_IES_other_direction.
```

```
(** ** Support for [CasariAnt IES]

   Now, we can stick our previously proved propositions
   together. By this, we get that [CasariAnt IES] is
   valid in our instantiated model [M].

   For this, we use classical logic in two points:
   - In order to apply contraposition via [NNPP] and
   - when we are applying
     [not_E_finitely_many_complement].
 *)
Proposition support_CasariAnt_IES :
  forall (s : state) (a : assignment),
    s, a |= <{CasariAnt IES}>.
Proof.
  intros s a i t H1 H2.
  apply support_CasariSuc_IES.

  apply NNPP.
  intros H3.
  eapply support_CasariImpl_IES_other_direction.
  -
    apply not_E_contains_all.
    exact H3.
  -
    apply not_E_finitely_many_complement.
    exact H3.
  -
    exact H2.
Qed.

Print Assumptions support_CasariAnt_IES.

(** ** Support for [Casari IES]

   We now conclude that we have indeed found a suitable
   counter-example. For this, we still need to define a
   suitable state. We would also need a concrete
   [assignment] but this can be done one the fly.

   [counter_state e] is a state that contains every odd
   number and every (even) number greater than [e]. By
   this, it contains at least one odd number and its
   complement can only contain infinitely many even
   numbers.
 *)
Local Program Definition counter_state (e : nat) : state :
  {|
```

```
Theorem not_support_valid_Casari_IES :
  ~ support_valid <{Casari IES}>.
Proof.
  intros H1.

  (**
     As [Casari IES] is an implication with conclusion
     [CasariSuc IES], we try to falsify this.
   *)
  eapply support_CasariSuc_IES_other_direction.
  -
    apply counter_state_contains_all_odds.
  -
    apply counter_state_contains_all_ltb.
  -
    eapply H1.
    +
      reflexivity.
    +
      fold support.
      apply support_CasariAnt_IES.

  (**
     We still need to instantiate some existential
     variables.
   *)
  Unshelve.
  exact (fun _ => 25). (* any variable [assignment] *)
  exact 24. (* concrete instance of [counter_state] *)
Qed.

Print Assumptions not_support_valid_Casari_IES.
(*
    Axioms:
      classic : forall P : Prop, P \/ ~ P
 *)
```

3. Conclusion & Future Work

Conclusion & Future Work

Conclusion

- We provide a case study regarding the syntactic approach using arity types.
- We formalized theory regarding (bounded) InqFOL using various methods.
- We extended the Sequent Calculus by Litak/Sano by allowing more generic signatures.
- We provide large demonstration proofs.

Conclusion & Future Work

Conclusion

- We provide a case study regarding the syntactic approach using arity types.
- We formalized theory regarding (bounded) InqFOL using various methods.
- We extended the Sequent Calculus by Litak/Sano by allowing more generic signatures.
- We provide large demonstration proofs.

Future Work

- Formalize the admissability of the rule of Cut within the Sequent Calculus.
- Implement a proof of Completeness for the Sequent Calculus.
- Derive the cardinality formulae within the Sequent Calculus.
- ...

References

References

[CG22] I. Ciardelli and G. Grilletti. “Coherence in inquisitive first-order logic”. en. In: *Annals of Pure and Applied Logic* 173.9 (Oct. 2022), p. 103155. DOI: 10.1016/j.apal.2022.103155.

[Cia22] I. Ciardelli. *Inquisitive Logic: Consequence and Inference in the Realm of Questions*. en. Vol. 60. Trends in Logic. Cham: Springer International Publishing, 2022. DOI: 10.1007/978-3-031-09706-5.

[dBr72] N. G de Bruijn. “Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem”. In: *Indagationes Mathematicae (Proceedings)* 75.5 (Jan. 1972), pp. 381–392. DOI: 10.1016/1385-7258(72)90034-0.

[LS] T. Litak and K. Sano. “Bounded inquistive logics: Sequent calculi and schematic validity”. en. In: ().

[STS] S. Schäfer, T. Tebbi, and G. Smolka. “Autosubst: Reasoning with de Bruijn Terms and Parallel Substitution”. en. In: ().