Propositional Logic is insufficient

\[ A \]

**ALL PERSONS ARE HAPPY**
Propositional Logic is insufficient

\[ A \quad \text{ALL PERSONS ARE HAPPY} \]
\[ B \quad \text{PAT IS A PERSON} \]
Propositional Logic is insufficient

\[ A \quad \text{ALL PERSONS ARE HAPPY} \]

\[ B \quad \text{PAT IS A PERSON} \]

\[ ? \quad \text{PAT IS HAPPY} \]

Propositional logic lacks possibility to talk about individuals
In particular, need to model objects, attributes, associations, etc.
Propositional Logic is insufficient

\[ A \quad \text{ALL PERSONS ARE HAPPY} \]
\[ B \quad \text{PAT IS A PERSON} \]
\[ ? \quad \text{PAT IS HAPPY} \]

Propositional logic lacks possibility to talk about individuals
In particular, need to model objects, attributes, associations, etc.

⇒ First-Order Logic (FOL) with Types
First-Order Logic

First-Order Formulas

First-Order Models

First-Order Sequent Calculus

\[ I, \models \]

\[ \vdash \]
OO Type Hierarchy

- Finite set $\mathcal{T}$ of static types, subtype relation $\subseteq$,
- Dynamic types $\mathcal{T}_d \subseteq \mathcal{T}$, where $\top \in \mathcal{T}_d$
- Abstract types $\mathcal{T}_a \subseteq \mathcal{T}$, where $\bot \in \mathcal{T}_a$
- $\mathcal{T}_d \cap \mathcal{T}_a = \emptyset$, $\mathcal{T}_d \cup \mathcal{T}_a = \mathcal{T}$, $\bot \subseteq z \subseteq \top$ for all $z \in \mathcal{T}$
Signature of Typed First-Order Logic

Given type hierarchy $(\mathcal{T}, \mathcal{T}_d, \mathcal{T}_a, \sqsubseteq)$, let $\mathcal{T}_q := \mathcal{T} \setminus \{\bot\}$

Signature $\Sigma = (V, P, F, \alpha)$
Signature of Typed First-Order Logic

Given type hierarchy \((\mathcal{T}, \mathcal{T}_d, \mathcal{T}_a, \sqsubseteq)\), let \(\mathcal{T}_q := \mathcal{T} \setminus \{\bot\}\)

**Signature** \(\Sigma = (\mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha)\)

**Variable Symbols** \(\mathcal{V} = \{x_i \mid i \in \mathbb{N}\}\)

**Predicate Symbols** \(\mathcal{P} = \{p_i \mid i \in \mathbb{N}\}\)

**Function Symbols** \(\mathcal{F} = \{f_i \mid i \in \mathbb{N}\}\)
Signature of Typed First-Order Logic

Given type hierarchy \((\mathcal{T}, \mathcal{T}_d, \mathcal{T}_a, \sqsubseteq)\), let \(\mathcal{T}_q := \mathcal{T} \setminus \{\bot\}\)

**Signature** \(\Sigma = (\mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha)\)

**Variable Symbols** \(\mathcal{V} = \{x_i \mid i \in \mathbb{N}\}\)

**Predicate Symbols** \(\mathcal{P} = \{p_i \mid i \in \mathbb{N}\}\)

**Function Symbols** \(\mathcal{F} = \{f_i \mid i \in \mathbb{N}\}\)

**Typing function** \(\alpha\) for all symbols:

\(\alpha(x) \in \mathcal{T}_q\) for all \(x \in \mathcal{V}\)

We write \(x: z\) instead of \(\alpha(x) = z\) (in Java: “\(z \; t;\)”)

\(\alpha(p) \in \mathcal{T}_q^*\) for all \(p \in \mathcal{P}\)

We write \(p: z_1, \ldots, z_r\) instead of \(\alpha(p) = (z_1, \ldots, z_r)\)

\(\alpha(f) \in \mathcal{T}_q^* \times \mathcal{T}_q\) for all \(f \in \mathcal{F}\)

We write \(f: z_1, \ldots, z_r \rightarrow z\) instead of \(\alpha(f) = ((z_1, \ldots, z_r), z)\)

\(r = 0\) ok, **No overloading of variables, functions, predicates!**
Special Signature Symbols

An **Equality** symbol $\doteq$ in $P$, with typing $\doteq : \top, \top$

A **type predicate** symbol $\in_z$ in $P$ for each $z \in \mathcal{T}_q$.
with typing $\in_z : \top$

**Type cast** symbol $(z)$ in $F$ for each $z \in \mathcal{T}_q$,
with typing $(z) : \top, z$
First-Order Signature Example

*Sticks and stones may break your bones, but flowers will never hurt*
First-Order Signature Example

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**
\[ T_d = \{ \text{Stick}, \text{Stone}, \text{Flower} \}, \quad T_a = \{ \text{Weapon}, \text{Any} \} \]

Stick, Stone ⊑ Weapon ⊑ Any, Flower ⊑ Any

**Predicates**
\[ P = \{ \text{hurts} : \text{Any} \} \]

**Functions**
\[ F = \{ \text{stick} : \rightarrow \text{Stick}, \text{stone} : \rightarrow \text{Stone}, r : \rightarrow \text{Flower} \} \]

Function with empty argument list: constant
**First-Order Signature Example**

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**
\[
\mathcal{T}_d = \{\text{Stick, Stone, Flower}\}, \quad \mathcal{T}_a = \{\text{Weapon, Any}\}
\]

Stick, Stone ⊑ Weapon ⊑ Any, Flower ⊑ Any

**Predicates**
\[
P = \{\text{hurts} : \text{Any}\}
\]

**Functions**
\[
F = \{\text{stick} : \rightarrow \text{Stick}, \text{stone} : \rightarrow \text{Stone}, r : \rightarrow \text{Flower}\}
\]

Function with empty argument list: constant

Terms of First-Order Logic

Given signature \((V, P, F, \alpha)\)

**Terms:** Set \(\text{Term}_z\) of terms of type \(z\), one for each **static type** \(z \in \mathcal{T}\)

- \(x\) is term of type \(z\) for each variable \(x : z\)

- \(f(t_1, \ldots, t_r)\) is term of type \(z\) for each function symbol \(f : z_1, \ldots, z_r \rightarrow z\) and terms \(t_i\) of type \(z'_i \sqsubseteq z_i\) for \(1 \leq i \leq r\)

If \(f\) is constant \((r = 0)\) we write \(f\) instead of \(f()\)
Terms of First-Order Logic

Given signature \((V, P, F, \alpha)\)

**Terms:** Set \(\text{Term}_z\) of terms of type \(z\), one for each **static type** \(z \in T\)

- \(x\) is term of type \(z\) for each variable \(x : z\)
- \(f(t_1, \ldots, t_r)\) is term of type \(z\) for each function symbol \(f : z_1, \ldots, z_r \rightarrow z\) and terms \(t_i\) of type \(z'_i \sqsubseteq z_i\) for \(1 \leq i \leq r\)
  If \(f\) is constant \((r = 0)\) we write \(f\) instead of \(f()\)

**Example:**

\(T_d = \{\text{Car, Person, } \top\}\) where \(\text{Person} \sqsubseteq \top, \text{Car} \sqsubseteq \top\)

\(F = \{\text{owner : Car} \rightarrow \text{Person, pat} : \rightarrow \text{Person, herbie} : \rightarrow \text{Car}\}, \ x : \text{Car}\)

**Terms:** herbie, owner(herbie), owner((Car)pat) (!), owner(x)

**Non-terms:** Car, owner(pat), owner(((Person)herbie)
Formulas of First-Order Logic

First-Order Formulas: Set $For$ of (first-order) formulas

- $p(t_1, \ldots, t_r)$ is an **atomic** formula for predicate symbol $p : z_1, \ldots, z_r$ and terms $t_i$ of type $z'_i \sqsubseteq z_i$ for $1 \leq i \leq r$

- **Truth constants**, **connectives** as in propositional logic

- If $x$ is any variable, $\phi$ a formula, then $\forall x. \phi$ and $\exists x. \phi$ are formulas

  We call $\phi$ the **scope** of variable $x$. We say that $x$ is **bound** by the **quantifier** $\forall$ in $\forall x. \phi$ (similarly for $\exists x. \phi$)
Formulas of First-Order Logic

First-Order Formulas: Set $\mathit{For}$ of (first-order) formulas

- $p(t_1, \ldots, t_r)$ is an atomic formula for predicate symbol $p : z_1, \ldots, z_r$ and terms $t_i$ of type $z'_i \sqsubseteq z_i$ for $1 \leq i \leq r$

- Truth constants, connectives as in propositional logic

If $x$ is any variable, $\phi$ a formula, then $\forall x. \phi$ and $\exists x. \phi$ are formulas

We call $\phi$ the scope of variable $x$. We say that $x$ is bound by the quantifier $\forall$ in $\forall x. \phi$ (similarly for $\exists x. \phi$)

Bound variables in quantified formulas are analogous to local variables/formal parameters in programs

Use pathentheses and usual precedence rules to avoid syntactic ambiguity
First-Order Syntax Example

Sticks and stones may break your bones, but flowers will never hurt

Types
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]
Stick, Stone \sqsubseteq \text{Weapon} \sqsubseteq \text{Any}, \text{Flower} \sqsubseteq \text{Any}

Predicates
\[ P = \{ \text{hurts : Any} \} \]

Functions
\[ F = \{ \text{stick : } \to \text{ Stick, stone : } \to \text{ Stone, r : } \to \text{ Flower} \} \]

Variables
\[ V = \{ x : \text{ Weapon}, y : \text{ Flower} \} \]

Examples:
First-Order Syntax Example

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**

\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]

Stick, Stone \sqsubseteq Weapon \sqsubseteq Any, Flower \sqsubseteq Any

**Predicates**

\[ P = \{ \text{hurts : Any} \} \]

**Functions**

\[ F = \{ \text{stick : } \rightarrow \text{ Stick, stone : } \rightarrow \text{ Stone, r : } \rightarrow \text{ Flower} \} \]

**Variables**

\[ V = \{ x : \text{ Weapon, y : Flower} \} \]

**Examples:**

\[ \forall x . \text{hurts}(x) \quad \& \quad \forall y . \neg\text{hurts}(y) \]

We sometimes write the type of quantified variables explicitly.
First-Order Syntax Example

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**
\[ T_d \] = \{Stick, Stone, Flower\}, \quad \[ T_a \] = \{Weapon, Any\}
Stick, Stone ⊆ Weapon ⊆ Any, Flower ⊆ Any

**Predicates**
\[ P \] = \{hurts : Any\}

**Functions**
\[ F \] = \{stick : → Stick, stone : → Stone, r : → Flower\}

**Variables**
\[ V \] = \{x : Weapon, y : Flower\}

**Examples:**
\[ \forall x : Weapon . hurts(x) \land \forall y : Flower . !hurts(y) \]
Sticks and stones may break your bones, but flowers will never hurt

Types
\[ T_d = \{\text{Stick}, \text{Stone}, \text{Flower}\}, \quad T_a = \{\text{Weapon}, \text{Any}\} \]
Stick, Stone \sqsubseteq \text{Weapon} \sqsubseteq \text{Any}, \text{Flower} \sqsubseteq \text{Any}

Predicates
\[ P = \{\text{hurts} : \text{Any}\} \]

Functions
\[ F = \{\text{stick} : \rightarrow \text{Stick}, \text{stone} : \rightarrow \text{Stone}, r : \rightarrow \text{Flower}\} \]

Variables
\[ V = \{x : \text{Weapon}, y : \text{Flower}\} \]

Examples:
\[ \forall x : \text{Weapon}. \text{hurts}(x) \land \forall y : \text{Flower}. \lnot \text{hurts}(y) \]
\[ \text{hurts}(r) \rightarrow \exists y. \text{hurts}(y) \]
Semantics of First-Order Logic
Semantics of First-Order Logic

A **model** of FOL is a triple \( M = (D, \delta, I) \) where

- \( D \) is the **universe** or **domain**
  Contains “objects” and “values”
- \( \delta \) is a **dynamic typing** function \( \delta : D \rightarrow T_d \)
  Each domain element has dynamic (“runtime”) type
- \( I \) is an **interpretation** of the function and predicate symbols s.t.
  - If \( p : z_1, \ldots, z_r \in P \), then \( I(p) \subseteq D^{z_1} \times \cdots \times D^{z_r} \)
  - If \( f : z_1, \ldots, z_r \rightarrow z \in F \), then \( I(f) : D^{z_1} \times \cdots \times D^{z_r} \rightarrow D^z \)

Moreover, let \( D^z = \{ d \in D \mid \delta(d) \sqsubseteq z \} \)

(the **domain elements of type** \( z \)).

The dynamic types \( z \in T_d \) **must be non-empty**: \( D^z \neq \emptyset \)
Semantics of Special Symbols

**Equality symbol** \( \equiv \) in \( P \), with typing \( \equiv: \top, \top \)

**Semantics:** \( \mathcal{I}(\equiv) = \{(d, d) \mid d \in D\} \subseteq D^\top \times D^\top \)

“Referential Equality”
Semantics of Special Symbols

**Equality symbol** $\doteq$ in $P$, with typing $\doteq : \top, \top$

**Semantics:** $\mathcal{I}(\doteq) = \{(d, d) \mid d \in \mathcal{D}\} \subseteq \mathcal{D}^\top \times \mathcal{D}^\top$

“Referential Equality”

**Type predicate symbol** $\sqsubseteq$ in $P$ for each $z \in \mathcal{T}_q$, with typing $\sqsubseteq : \top$

**Semantics:** $\mathcal{I}(\sqsubseteq) = \mathcal{D}^z \subseteq \mathcal{D}^\top$
Semantics of Special Symbols

**Equality symbol** \( \hat{=} \) in \( P \), with typing \( \hat{=} : \top, \top \)

**Semantics:** \( I(\hat{=}) = \{(d, d) \mid d \in D\} \subseteq D\top \times D\top \)

“Referential Equality”

**Type predicate symbol** \( \sqsubseteq \) in \( P \) for each \( z \in Tq \), with typing \( \sqsubseteq : \top \)

**Semantics:** \( I(\sqsubseteq) = D^z \subseteq D\top \)

**Type cast symbol** \( (z) \) in \( F \) for each \( z \in Tq \), with typing \( (z) : \top, z \)

**Semantics:** \( I((z)) \) is a function such that

\[
I((z))(x) = \begin{cases} 
  x & \text{if } \delta(x) \sqsubseteq z \\
  d & \text{otherwise}
\end{cases}
\]

with \( d \) an arbitrary but fixed element of \( D^z \)
Semantics of First-Order Logic: Example

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]

**Predicates**
\[ P = \{ \text{hurts : Any} \} \]

**Functions**
\[ F = \{ \text{stick : } \rightarrow \text{ Stick, stone : } \rightarrow \text{ Stone, r : } \rightarrow \text{ Flower} \} \]

**Variables**
\[ V = \{ x : \text{ Weapon, y : Flower} \} \]

One of (infinitely) many possible models:
Semantics of First-Order Logic: Example

Sticks and stones may break your bones, but flowers will never hurt

Types
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]

Stick, Stone ⊑ Weapon ⊑ Any, Flower ⊑ Any

Predicates
\[ P = \{ \text{hurts} : \text{Any} \} \]

Functions
\[ F = \{ \text{stick} : \rightarrow \text{Stick}, \text{stone} : \rightarrow \text{Stone}, r : \rightarrow \text{Flower} \} \]

Variables
\[ V = \{ x : \text{Weapon}, y : \text{Flower} \} \]

One of (infinitely) many possible models:

Domain
\[ D = \{ o_1, o_2, o_3, o_4 \} \]
Semantics of First-Order Logic: Example

*Sticks and stones may break your bones, but flowers will never hurt*

Types

\[ T_d = \{\text{Stick, Stone, Flower}\}, \quad T_a = \{\text{Weapon, Any}\} \]

\( \text{Stick, Stone} \sqsubseteq \text{Weapon} \sqsubseteq \text{Any}, \text{Flower} \sqsubseteq \text{Any} \)

Predicates

\[ P = \{\text{hurts} : \text{Any}\} \]

Functions

\[ F = \{\text{stick} : \rightarrow \text{Stick}, \text{stone} : \rightarrow \text{Stone}, r : \rightarrow \text{Flower}\} \]

Variables

\[ V = \{x : \text{Weapon}, y : \text{Flower}\} \]

One of (infinitely) many possible models:

Domain

\[ D = \{o_1, o_2, o_3, o_4\} \]

Typing

\[ \delta(o_1) = \delta(o_4) = \text{Stick}, \quad \delta(o_2) = \text{Stone}, \quad \delta(o_3) = \text{Flower} \]

\[ D^{\text{Stick}} = \{o_1, o_4\}, \quad D^{\text{Stone}} = \{o_2\}, \quad D^{\text{Flower}} = \{o_3\}, \quad D^{\text{Any}} = \{o_1, o_2, o_3, o_4\} \]
Semantics of First-Order Logic: Example

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]
 Stick, Stone ⊑ Weapon ⊑ Any, Flower ⊑ Any

**Predicates**
\[ P = \{ \text{hurts : Any} \} \]

**Functions**
\[ F = \{ \text{stick} : \rightarrow \text{Stick}, \text{stone} : \rightarrow \text{Stone}, r : \rightarrow \text{Flower} \} \]

**Variables**
\[ V = \{ x : \text{Weapon}, y : \text{Flower} \} \]

One of (infinitely) many possible models:

**Domain**
\[ D = \{ o_1, o_2, o_3, o_4 \} \]

**Typing**
\[ \delta(o_1) = \delta(o_4) = \text{Stick}, \quad \delta(o_2) = \text{Stone}, \quad \delta(o_3) = \text{Flower} \]
\[ D^\text{Stick} = \{ o_1, o_4 \}, \quad D^\text{Stone} = \{ o_2 \}, \quad D^\text{Flower} = \{ o_3 \}, \quad D^\text{Any} = \{ o_1, o_2, o_3, o_4 \} \]

**Interpretation**
\[ I(\text{hurts}) = \{ o_1, o_2, o_4 \} \]
\[ I(\text{stick}) = o_1, \quad I(\text{stone}) = o_2, \quad I(r) = o_3 \]
Assigning meaning to variables

Let \( x \) be variable of static type \( z \)

A Variable Assignment \( \beta \) maps \( x \) to an element of \( D^z \)
Assigning meaning to variables

Let $x$ be variable of static type $z$

A Variable Assignment $\beta$ maps $x$ to an element of $\mathcal{D}^z$

Assigning meaning to terms: a mapping $val_{\mathcal{M},\beta}$ from $\text{Term}_z(t)$ to $\mathcal{D}^z$ (depending on model $\mathcal{M}$ and variable assignment $\beta$) such that

\[
\begin{align*}
val_{\mathcal{M},\beta}(x) &= \beta(x) \quad \text{(element in $\mathcal{D}^z$, where $x$ has type $z$)} \\
val_{\mathcal{M},\beta}(f(t_1, \ldots, t_r)) &= \mathcal{I}(f)(\val_{\mathcal{M},\beta}(t_1), \ldots, \val_{\mathcal{M},\beta}(t_r))
\end{align*}
\]
Assigning meaning to variables

Let \( x \) be variable of static type \( z \)

A Variable Assignment \( \beta \) maps \( x \) to an element of \( D^z \)

Assigning meaning to terms: a mapping \( \text{val}_{\mathcal{M}, \beta} \) from \( \text{Term}_z(t) \) to \( D^z \) (depending on model \( \mathcal{M} \) and variable assignment \( \beta \)) such that

\[
\begin{align*}
\text{val}_{\mathcal{M}, \beta}(x) &= \beta(x) \quad &\text{(element in } D^z, \text{ where } x \text{ has type } z) \\
\text{val}_{\mathcal{M}, \beta}(f(t_1, \ldots, t_r)) &= I(f)(\text{val}_{\mathcal{M}, \beta}(t_1), \ldots, \text{val}_{\mathcal{M}, \beta}(t_r))
\end{align*}
\]

Modified variable assignment:

For \( d \in D^z \) let \( \beta^d_y(x) := \begin{cases} 
\beta(x) & \text{if } x \neq y \\
d & \text{if } x = y 
\end{cases} \)
Assigning meaning to formulas

Validity relation: \( M, \beta \models \phi \) for \( \phi \in For \)

- \( M, \beta \models p(t_1, \ldots, t_r) \) iff \( (\text{val}_{M, \beta}(t_1), \ldots, \text{val}_{M, \beta}(t_r)) \in \mathcal{I}(p) \)

- \( M, \beta \models \phi \land \psi \) iff \( M, \beta \models \phi \) and \( M, \beta \models \psi \)

- \( \ldots \)

- \( M, \beta \models \forall x. \phi \) iff \( M, \beta^d_x \models \phi \) for all \( d \in D^z \) where the type of \( x \) is \( z \)

- \( M, \beta \models \exists x. \phi \) iff \( M, \beta^d_x \models \phi \) for at least one \( d \in D^z \) where the type of \( x \) is \( z \)
Semantics of First-Order Logic: Example

*Sticks and stones may break your bones, but flowers will never hurt*

**Types**

\[ \mathcal{T}_d = \{\text{Stick, Stone, Flower}\}, \quad \mathcal{T}_a = \{\text{Weapon, Any}\} \]

Stick, Stone ⊑ Weapon ⊑ Any, Flower ⊑ Any

**Predicates**

\[ \mathcal{P} = \{\text{hurts : Any}\} \]

**Functions**

\[ \mathcal{F} = \{\text{stick : Stick, stone : Stone, } r : \text{Flower}\} \]

**Variables**

\[ \mathcal{V} = \{x : \text{Weapon, } y : \text{Flower}\} \]

In our previous model \( \mathcal{M} \):

\[ \mathcal{D}_{\text{Stick}} = \{o_1, o_4\}, \quad \mathcal{D}_{\text{Stone}} = \{o_2\}, \quad \mathcal{D}_{\text{Flower}} = \{o_3\} \]

\[ \mathcal{D}_{\text{Weapon}} = \{o_1, o_2, o_4\}, \quad \mathcal{I}(\text{hurts}) = \{o_1, o_2, o_4\} \subseteq \mathcal{D}_{\text{Any}} \]

Evaluate these formulas: \( \exists x. \text{hurts}(x), \quad \forall x. \text{hurts}(x), \quad \exists y. \text{hurts}(y) \)
Let \( \beta \) be arbitrary.

\[ \mathcal{M}, \beta \models \exists x : \text{Weapon} . \text{hurts}(x) \quad \text{iff} \]
Semantics of First-Order Logic: Evaluation Example

Let $\beta$ be arbitrary.

$\mathcal{M}, \beta \models \exists x : \text{Weapon} \cdot \text{hurts}(x)$  iff  
There exists $d \in \mathcal{D}^{\text{Weapon}}$ such that $\mathcal{M}, \beta^d_x \models \text{hurts}(x)$  if

Semantic Rule

$\mathcal{M}, \beta \models \exists x . \phi$  iff  $\mathcal{M}, \beta^d_x \models \phi$ for at least one $d \in \mathcal{D}^z$
where the type of $x$ is $z$

Information from model $(\mathcal{D}, \delta, \mathcal{I})$
Let $\beta$ be arbitrary.

$\mathcal{M}, \beta \models \exists x : \text{Weapon} \cdot \text{hurts}(x) \quad \text{iff} \quad \exists d \in D^{\text{Weapon}} \text{ such that } \mathcal{M}, \beta^d \models \text{hurts}(x)$

There exists $d \in D^{\text{Weapon}}$ such that $\mathcal{M}, \beta^d \models \text{hurts}(x)$ if

$\mathcal{M}, \beta^{o_1} \models \text{hurts}(x) \quad \text{iff} \quad \mathcal{M}, \beta^{o_2} \models \text{hurts}(x)$

Semantic Rule

Information from model $(D, \delta, I)$

$D^{\text{Weapon}} = \{ o_1, o_2, o_4 \}$
Let $\beta$ be arbitrary.

$\mathcal{M}, \beta \models \exists x : \text{Weapon}. \text{hurts}(x)$ iff

There exists $d \in D^{\text{Weapon}}$ such that $\mathcal{M}, \beta^d_x \models \text{hurts}(x)$ if

$\mathcal{M}, \beta^o_x \models \text{hurts}(x)$ iff

$\text{val}_{\mathcal{M}, \beta^o_x}(x) \in I(\text{hurts})$

**Semantic Rule**

$\mathcal{M}, \beta \models p(t_1, \ldots, t_r)$ iff $(\text{val}_{\mathcal{M}, \beta}(t_1), \ldots, \text{val}_{\mathcal{M}, \beta}(t_r)) \in I(p)$

**Information from model** $(\mathcal{D}, \delta, I)$
Semantics of First-Order Logic: Evaluation Example

Let $\beta$ be arbitrary.

$\mathcal{M}, \beta \models \exists x : \text{Weapon} \cdot \text{hurts}(x)$ \iff

There exists $d \in \mathcal{D}^\text{Weapon}$ such that $\mathcal{M}, \beta_d^x \models \text{hurts}(x)$ if

$\mathcal{M}, \beta_{x}^{o_1} \models \text{hurts}(x)$ \iff

$\text{val}_{\mathcal{M}, \beta_{x}^{o_1}}(x) \in \mathcal{I}(\text{hurts})$

since $\text{val}_{\mathcal{M}, \beta_{x}^{o_1}}(x) = \beta_{x}^{o_1}(x) = o_1$ \iff

Semantic Rule

$\text{val}_{\mathcal{M}, \beta}(x) = \beta(x)$, \quad $\beta_d^x(x) := \begin{cases} \beta(x) & x \neq y \\ d & x = y \end{cases}$

Information from model \((\mathcal{D}, \delta, \mathcal{I})\)
Let $\beta$ be arbitrary.

$\mathcal{M}, \beta \models \exists x : \text{Weapon}. \text{hurts}(x)$     iff

There exists $d \in D^{\text{Weapon}}$ such that $\mathcal{M}, \beta^d_x \models \text{hurts}(x)$     if

$\mathcal{M}, \beta^o_{x1} \models \text{hurts}(x)$     iff

$\text{val}_{\mathcal{M}, \beta^o_{x1}}(x) \in I(\text{hurts})$

since     $\text{val}_{\mathcal{M}, \beta^o_{x1}}(x) = \beta^o_{x1}(x) = o_1$     iff

$o_1 \in I(\text{hurts}) = \{o_1, o_2, o_4\}$

Semantic Rule

Information from model $(D, \delta, I)$

$I(\text{hurts}) = \{o_1, o_2, o_4\}$
Semantics of First-Order Logic: Evaluation Example

Let \( \beta \) be arbitrary.

\[ M, \beta \models \exists x : \text{Weapon} . \text{hurts}(x) \quad \text{iff} \]

There exists \( d \in D^{\text{Weapon}} \) such that \( M, \beta_d^x \models \text{hurts}(x) \) if

\[ M, \beta_{x}^{o_1} \models \text{hurts}(x) \quad \text{iff} \]

\( \text{val}_{M, \beta_{x}^{o_1}}(x) \in I(\text{hurts}) \)

since \( \text{val}_{M, \beta_{x}^{o_1}}(x) = \beta_{x}^{o_1}(x) = o_1 \) \( \text{iff} \)

\( o_1 \in I(\text{hurts}) = \{ o_1, o_2, o_4 \} \) \quad \text{ok!} \)

Semantic Rule

Information from model \((D, \delta, I)\)
First-Order Semantic Notions

Satisfiability, truth, and validity

\[ M, \beta \models \phi \]  \quad (\phi \text{ is satisfiable})

\[ M \models \phi \text{ iff for all } \beta : \ M, \beta \models \phi \]  \quad (\phi \text{ is true in } M)

\[ \models \phi \text{ iff for all } M : \ M \models \phi \]  \quad (\phi \text{ is valid})

Formula containing only variables in scope of a quantifier is closed
Closed formulas that are satisfiable are also true: only one notion

From now on only closed formulas are considered.
First-Order Logic Example

Types
\[ T_d = \{\text{Stick, Stone, Flower}\}, \quad T_a = \{\text{Weapon, Any}\}\]
Stick, Stone \sqsubseteq \text{Weapon} \sqsubseteq \text{Any}, \text{Flower} \sqsubseteq \text{Any}

Predicates
\[ P = \{\text{hurts : Any}\}\]

Variables
\[ V = \{x : \text{Weapon}, y : \text{Flower}\}\]
First-Order Logic Example

Types \( \mathcal{T}_d = \{\text{Stick, Stone, Flower}\}, \quad \mathcal{T}_a = \{\text{Weapon, Any}\} \)
Stick, Stone \( \sqsubseteq \) Weapon \( \sqsubseteq \) Any, Flower \( \sqsubseteq \) Any

Predicates \( P = \{\text{hurts : Any}\} \)

Variables \( V = \{x : \text{Weapon}, y : \text{Flower}\} \)

\[ \forall x : \text{Weapon} . \text{hurts}(x) \quad \& \quad \forall y : \text{Flower} . \neg \text{hurts}(y) \]

Satisfiable? True? Valid?
First-Order Logic Example

Types
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]
Stick, Stone \sqsubseteq \text{Weapon} \sqsubseteq \text{Any}, \text{Flower} \sqsubseteq \text{Any}

Predicates
\[ P = \{ \text{hurts : Any} \} \]

Variables
\[ V = \{ x : \text{Weapon}, y : \text{Flower} \} \]

\[ \forall x : \text{Weapon}. \text{hurts}(x) \quad \& \quad \forall y : \text{Flower}. \neg \text{hurts}(y) \]

Satisfiable? True? Valid?
Model:
\[ \mathcal{D} = \{ o_1, o_2 \}, \quad \delta(o_1) = \text{Stone}, \quad \delta(o_2) = \text{Flower} \]
\[ \mathcal{I}(\text{hurts}) = \{ o_1 \} \]
First-Order Logic Example

Types
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]
Stick, Stone \subseteq \text{Weapon, Any}, Flower \subseteq \text{Any}

Predicates
\[ P = \{ \text{hurts : Any} \} \]

Variables
\[ V = \{ x : \text{Weapon}, y : \text{Flower} \} \]

\[ \forall x : \text{Weapon} . \text{hurts}(x) \quad \& \quad \forall y : \text{Flower} . \lnot \text{hurts}(y) \]

Satisfiable? True? Valid?
Counter-model:
\[ D = \{ o_1, o_2 \}, \quad \delta(o_1) = \text{Stone}, \quad \delta(o_2) = \text{Flower} \]
\[ \mathcal{I}(\text{hurts}) = \{ \} \]
First-Order Logic Example

**Types**
\[ T_d = \{ \text{Stick, Stone, Flower} \}, \quad T_a = \{ \text{Weapon, Any} \} \]
Stick, Stone \sqsubseteq \text{Weapon} \sqsubseteq \text{Any}, \text{ Flower} \sqsubseteq \text{Any}

**Predicates**
\[ P = \{ \text{hurts : Any} \} \]

**Variables**
\[ V = \{ x : \text{Weapon}, y : \text{Flower} \} \]

\[
\forall x : \text{Weapon}. \text{hurts}(x) \quad \& \quad \forall y : \text{Flower}. \neg \text{hurts}(y)
\]

**Satisfiable? True? Valid?**

**Another Counter-model:**
\[ D = \{ o_1, o_2, o_3 \}, \quad \delta(o_1) = \text{Stone}, \quad \delta(o_2) = \delta(o_3) = \text{Flower} \]
\[ I(\text{hurts}) = \{ o_1, o_3 \} \]
Untyped First-Order Logic

Standard FOL (as in most logic textbooks is untyped [single typed])

Obtained as special case of typed signature:

\[ T_d = \{ \top \}, \quad T_a = \{ \bot \} \]

Hence, \( D = D^\top \neq \emptyset \), \( \delta(d) = \top \) for all \( d \in D \)

All variables, predicate and function symbols declared on \( \top \)

Don’t need type information of variables (omit)

Only arity in signature of function/predicate symbols matters
Untyped First-Order Logic

Standard FOL (as in most logic textbooks is untyped [single typed])

Obtained as special case of typed signature:

\[ T_d = \{ \top \}, \quad T_a = \{ \bot \} \]

Hence, \( D = D^\top \neq \emptyset \), \( \delta(d) = \top \) for all \( d \in D \)

All variables, predicate and function symbols declared on \( \top \)

Don’t need type information of variables (omit)

Only arity in signature of function/predicate symbols matters

Example: \( P = \{ \text{person}/1, \text{happy}/1 \}, \quad F = \{ \text{pat}/0 \} \)
Untyped First-Order Logic

Standard FOL (as in most logic textbooks is untyped [single typed])

Obtained as special case of typed signature:

\[ T_d = \{ \top \}, \quad T_a = \{ \bot \} \]

Hence, \[ D = D^\top \neq \emptyset, \quad \delta(d) = \top \text{ for all } d \in D \]

All variables, predicate and function symbols declared on \[ \top \]

Don’t need type information of variables (omit)

Only arity in signature of function/predicate symbols matters

Example: \[ P = \{ \text{person} / 1, \text{happy} / 1 \}, \quad F = \{ \text{pat} / 0 \} \]

\[ \forall x. (\text{person}(x) \rightarrow \text{happy}(x)) \quad \text{ALL PERSONS ARE HAPPY} \]
Untyped First-Order Logic

Standard FOL (as in most logic textbooks is untyped [single typed])

 Obtained as special case of typed signature:

\[ T_d = \{ \top \}, \quad T_a = \{ \bot \} \]

Hence, \( D = D^\top \neq \emptyset \), \( \delta(d) = \top \) for all \( d \in D \)

All variables, predicate and function symbols declared on \( \top \)

Don’t need type information of variables (omit)

Only arity in signature of function/predicate symbols matters

Example: \( P = \{ \text{person}/1, \text{happy}/1 \} \), \( F = \{ \text{pat}/0 \} \)

\[ \forall x . (\text{person}(x) \rightarrow \text{happy}(x)) \]

ALL PERSONS ARE HAPPY

person(pat) \quad \text{PAT IS A PERSON}
Untyped First-Order Logic

Standard FOL (as in most logic textbooks is untyped [single typed])

Obtained as special case of typed signature:

\( T_d = \{ \top \} \), \( T_a = \{ \bot \} \)

Hence, \( D = D^\top \neq \emptyset \), \( \delta(d) = \top \) for all \( d \in D \)

All variables, predicate and function symbols declared on \( \top \)

Don’t need type information of variables (omit)

Only arity in signature of function/predicate symbols matters

Example: \( P = \{ \text{person}/1, \text{happy}/1 \} \), \( F = \{ \text{pat}/0 \} \)

\( \forall x . (\text{person}(x) \rightarrow \text{happy}(x)) \) \hspace{1cm} \text{ALL PERSONS ARE HAPPY}

\( \text{person(pat)} \) \hspace{2cm} \text{PAT IS A PERSON}

\( \text{happy(pat)} \) \hspace{2cm} \text{PAT IS HAPPY}
Certain symbols should have “standard” meaning in all interpretations

So far: \( \doteq, \in \mathbb{Z}, (z) \)

For certain types we also fix domain and dynamic typing:

\[
D^{\text{int}} = \{ d \in D \mid \delta(d) = \text{int} \} = \mathbb{Z}
\]

These types appear between \( \bot \) and \( \top \), uncomparable to others

Examples of types, function/predicate symbols with fixed meaning

\( \mathcal{I}(17) \) should be always 17, not e.g. towel

\textbf{int} KeY can switch between JAVA 32-bit integers and \( \mathbb{Z} \)
but in FOL always math integers \( \mathcal{I}(+) = +\mathbb{Z}, \ \mathcal{I}(\ast) = \ast\mathbb{Z} \), ...

\textbf{boolean}
Some Predefined Symbols in KeY FO Logic

Types

int, short, byte, boolean with standard meaning

All classes of current UML context diagram and Null

If $T$ is one of these types then also $Set(T)$, $Bag(T)$, $Sequence(T)$

Predicates on integer types with standard meaning

>$, <, \geq, \leq, \ldots$ (infix)

Functions and Constants with standard meaning

$+, -, \div, \mod, 0, 1, \ldots$

TRUE, FALSE

Notation for quantifiers, variables declared at quantifier symbol

$\forall$ Type Variable; Scope Formula
First-Order Problems in KeY Syntax: .key

\sorts { // types are called 'sorts'
    person; // one declaration per line, end with ';
}

\functions { // ResultType FctSymbol(ParType,..,ParType)
    int age(person); // 'int' predefined type
}

\predicates { // PredSymbol(ParType,..,ParType)
    parent(person,person);
}

\problem { // Goal formula
    \forall person son; \forall person father;
    (parent(father,son) -> age(father) > age(son)) }

Contents

- Overview of KeY
- UML and its semantics
- Introduction to OCL
- Specifying requirements with OCL
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
</table>

- $[t/t']\phi$ is result of replacing each occurrence of $t$ in $\phi$ with $t'$
- $s^z, t^{z'}$ and $t$ are arbitrary variable free terms
- $x$ and $s^z$ have static type $z$ and $t^{z'}$ has static type $z' \sqsubseteq z$
- $c^z$ new constant of type $z$ (does not occur in current proof branch)
- Equations can be reversed (by symmetry of equality)
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall ) [ \Gamma, \forall x.\phi, [x/t^{z'}] \phi \implies \Delta ]</td>
<td>[ \Gamma \implies [x/c^{z}] \phi, \Delta ]</td>
</tr>
<tr>
<td>[ \Gamma, \forall x.\phi \implies \Delta ]</td>
<td>[ \Gamma \implies \forall x.\phi, \Delta ]</td>
</tr>
</tbody>
</table>

- \([t/t']\phi\) is result of replacing each occurrence of \(t\) in \(\phi\) with \(t'\)
- \(s^{z}, t^{z'}\) and \(t\) are arbitrary variable free terms
- \(x\) and \(s^{z}\) have static type \(z\) and \(t^{z'}\) has static type \(z' \sqsubseteq z\)
- \(c^{z}\) new constant of type \(z\) (does not occur in current proof branch)
- Equations can be reversed (by symmetry of equality)
### Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall \Gamma, \forall x.\phi, [x/t^z'] \phi \implies \Delta$</td>
<td>$\Gamma \implies [x/c^z] \phi, \Delta$</td>
</tr>
<tr>
<td>$\Gamma, \forall x.\phi \implies \Delta$</td>
<td>$\Gamma \implies \forall x.\phi, \Delta$</td>
</tr>
<tr>
<td>$\exists \Gamma, [x/c^z] \phi \implies \Delta$</td>
<td>$\Gamma \implies [x/t^z'] \phi, \exists x.\phi, \Delta$</td>
</tr>
<tr>
<td>$\Gamma, \exists x.\phi \implies \Delta$</td>
<td>$\Gamma \implies \exists x.\phi, \Delta$</td>
</tr>
</tbody>
</table>

- $[t/t'] \phi$ is result of replacing each occurrence of $t$ in $\phi$ with $t'$
- $s^z, t^z'$ and $t$ are arbitrary variable free terms
- $x$ and $s^z$ have static type $z$ and $t^z'$ has static type $z' \sqsubseteq z$
- $c^z$ new constant of type $z$ (does not occur in current proof branch)
- Equations can be reversed (by symmetry of equality)
Sequent Calculus for FOL

<table>
<thead>
<tr>
<th>left side, antecedent</th>
<th>right side, succedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall )</td>
<td>( \exists )</td>
</tr>
<tr>
<td>( \Gamma, \forall x. \phi, [x/t'] \phi \implies \Delta )</td>
<td>( \Gamma \implies [x/c^z] \phi, \Delta )</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
<td>( \Gamma \implies \forall x. \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, \forall x. \phi \implies \Delta )</td>
<td>( \Gamma \implies [x/t'] \phi, \exists x. \phi, \Delta )</td>
</tr>
<tr>
<td>( \exists )</td>
<td>( \Gamma \implies \exists x. \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, [x/c^z] \phi \implies \Delta )</td>
<td>( \Gamma \implies [x/t'] \phi, \exists x. \phi, \Delta )</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
<td>( \Gamma \implies t \doteq t, \Delta )</td>
</tr>
<tr>
<td>( \Gamma )</td>
<td>( \Gamma \implies [s^z/t'] [s^z/t'] \psi \implies [s^z/t'] \phi, \Delta )</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
<td>( \Gamma \implies [s^z/t'] \phi, \Delta )</td>
</tr>
<tr>
<td>( \Gamma, s^z \doteq t', [s^z/t'] \psi \implies [s^z/t'] \phi, \Delta )</td>
<td>( \Gamma \implies s^z \doteq t', \psi \implies \phi, \Delta )</td>
</tr>
</tbody>
</table>

- \( [t/t'] \phi \) is result of replacing each occurrence of \( t \) in \( \phi \) with \( t' \)
- \( s^z, t'^z \) and \( t \) are arbitrary variable free terms
- \( x \) and \( s^z \) have static type \( z \) and \( t'^z \) has static type \( z' \sqsubseteq z \)
- \( c^z \) new constant of type \( z \) (does not occur in current proof branch)
- Equations can be reversed (by symmetry of equality)
A Simple Proof (Exercises p3.key)

\[ \exists x. \forall y. p(x, y) \implies \forall y. \exists x. p(x, y) \]  

Let static type of \( x \) and \( y \) be \( \top \)
A Simple Proof (Exercises p3.key)

\[ \forall y. p(c, y) \implies \forall y. \exists x. p(x, y) \]

\[ \exists x. \forall y. p(x, y) \implies \forall y. \exists x. p(x, y) \]

ex left: substitute new constant \( c \) of type \( \top \) for \( x \)
A Simple Proof (Exercises p3.key)

\[ \forall y. p(c, y) \implies \exists x. p(x, d) \]

\[ \forall y. p(c, y) \implies \forall y. \exists x. p(x, y) \]

\[ \exists x. \forall y. p(x, y) \implies \forall y. \exists x. p(x, y) \]

all right: substitute new constant \( d \) of type \( \top \) for \( y \)
A Simple Proof (Exercises p3.key)

\[
p(c, d), \forall y. p(c, y) \implies \exists x. p(x, d)
\]

\[
\forall y. p(c, y) \implies \exists x. p(x, d)
\]

\[
\forall y. p(c, y) \implies \forall y. \exists x. p(x, y)
\]

\[
\exists x. \forall y. p(x, y) \implies \forall y. \exists x. p(x, y)
\]

all left: free to substitute any term of type \(\top\) for \(y\), choose \(d\)
A Simple Proof (Exercises p3.key)

\[
p(c, d) \implies \exists x. p(x, d)
\]

\[
\forall y. p(c, y) \implies \exists x. p(x, d)
\]

\[
\forall y. p(c, y) \implies \forall y. \exists x. p(x, y)
\]

\[
\exists x. \forall y. p(x, y) \implies \forall y. \exists x. p(x, y)
\]

all left not needed anymore (hide)
A Simple Proof (Exercises p3.key)

\[
\begin{align*}
p(c, d) & \implies p(c, d), \ \exists x. p(x, y) \\
p(c, d) & \implies \exists x. p(x, d) \\
\forall y. p(c, y) & \implies \exists x. p(x, d) \\
\forall y. p(c, y) & \implies \forall y. \exists x. p(x, y) \\
\exists x. \forall y. p(x, y) & \implies \forall y. \exists x. p(x, y)
\end{align*}
\]

ex right: free to substitute any term of type \( \top \) for \( x \), choose \( c \)
A Simple Proof (Exercises p3.key)

\[ p(c,d) \implies p(c,d) \]

\[ p(c,d) \implies \exists x. p(x,d) \]

\[ \forall y. p(c,y) \implies \exists x. p(x,d) \]

\[ \forall y. p(c,y) \implies \forall y. \exists x. p(x,y) \]

\[ \exists x. \forall y. p(x,y) \implies \forall y. \exists x. p(x,y) \]

ex right not needed anymore (hide)
A Simple Proof (Exercises p3.key)

* 

\[ p(c, d) \implies p(c, d) \]

\[ p(c, d) \implies \exists x. p(x, d) \]

\[ \forall y. p(c, y) \implies \exists x. p(x, d) \]

\[ \forall y. p(c, y) \implies \forall y. \exists x. p(x, y) \]

\[ \exists x. \forall y. p(x, y) \implies \forall y. \exists x. p(x, y) \]

Close
Rules for Type Casts and Type Predicates

- **Type predicate** formulas $t \sqsubseteq z$
  true iff dynamic type $val_M(t)$ is subtype of $z$

- **Type cast** terms $(z)t$
  evaluates to $val_M(t)$ if cast succeeds, arb. element otherwise
Rules for Type Casts and Type Predicates

- **Type predicate formulas** \( t \sqsubseteq z \)
  true iff dynamic type \( val_M(t) \) is subtype of \( z \)

- **Type cast terms** \((z)t\)
  evaluates to \( val_M(t) \) if cast succeeds, arb. element otherwise

Typical rule:
Rules for Type Casts and Type Predicates

- **Type predicate** formulas $t \sqsubseteq z$
  true iff dynamic type $val_M(t)$ is subtype of $z$

- **Type cast** terms $(z)t$
  evaluates to $val_M(t)$ if cast succeeds, arb. element otherwise

Typical rule:
The dynamic type of a term must be typeable to its static type

$$\frac{\Gamma, t \sqsubseteq z \implies \Delta}{\Gamma \implies \Delta} \quad \text{TYPESTATIC}$$

$z$ static (declared) type of $t$

Expresses **type-safety** of typed first-order logic
Rules for Type Casts and Type Predicates

- **Type predicate formulas** $t \sqsubseteq z$
  true iff dynamic type $\text{val}_M(t)$ is subtype of $z$

- **Type cast terms** $(z)t$
  evaluates to $\text{val}_M(t)$ if cast succeeds, arb. element otherwise

Typical rule:

The dynamic type of a term must be typeable to its static type

$$\text{TYPE_Static} \quad \frac{\Gamma, t \sqsubseteq z \quad \Rightarrow \Delta}{\Gamma \quad \Rightarrow \Delta} \quad z \text{ static (declared) type of } t$$

Expresses **type-safety** of typed first-order logic

KeY first-order strategy applies suitable typing rules automatically
Sequent Proofs: Important Issues

- Rules are applied to top-most connective/quantifier
- \texttt{exLeft} and \texttt{allRight} substitute \textit{new} constant
- \texttt{exRight} and \texttt{allLeft} allow to substitute \textit{any} variable-free term
- Formulas that are not needed in remaining proof may be hidden
- All branches must be \textit{closed} with axiom
- There are many different possible proofs for a valid sequent
- KeY FO strategy applies all but \texttt{exRight} and \texttt{allLeft} automatically
Another Proof Example

**Types**
\[ \mathcal{T} = \{ \bot, \top \} \]

**Predicates**
\[ \text{PSym} = \{ p \}, \quad p : \top, \top \]

**Functions**
\[ \text{FSym} = \{ \} \]

\[
(\exists x . \exists y . p(x, y) \land \forall x . \neg p(x, x)) \implies \exists x . \exists y . (\neg x \equiv y)
\]

Intuitive Meaning? Satisfiable? True? Valid?

Demo

oclFol/rel.key