First Order Predicate Logic
Formal Semantics and Related Notions

Wolfgang Schreiner and Wolfgang Windsteiger
Wolfgang.(Schreiner|Windsteiger)@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University (JKU), Linz, Austria
http://www.risc.jku.at
Formal Semantics

Up to now, our presentation of predicate logic formulas, their manipulation and proving, was mainly based on the form (syntax) of the formulas; this leaves many questions open.

▶ Equivalence of formulas:
  ▶ What exactly does a formula mean, e.g., when do two syntactically different formulas express the same fact?

▶ Soundness and completeness of proving rules:
  ▶ Proving rules allow by only considering the form of formulas to judge that some formula is a consequence of some other formulas.
  ▶ But are the derived judgements really always true, i.e., are the rules really sound?
  ▶ Furthermore, can all true judgements be derived, i.e., are the rules also complete?

We will answer these questions by underpinning our previous presentation with a formal definition of the meaning (semantics) of formulas.
Formal Semantics

The meaning of a predicate logic formula depends on the following entities.

- **Domain** \(D\)
  - A non-empty set, the universe about which the formula talks.
  \[ D = \mathbb{N}. \]

- **Interpretation** \(I\) of all function and predicate symbols
  - **Constants**: For every constant \(c\), \(I(c)\) denotes an element of \(D\), i.e., \(I(c) \in D\).
  - **Functions**: For every function symbol \(f\) with arity \(n > 0\), \(I(f)\) denotes an \(n\)-ary function on \(D\), i.e., \(I(f) : D^n \rightarrow D\).
  - **Predicates**: For every predicate symbol \(p\) with arity \(n > 0\), \(I(p)\) denotes an \(n\)-ary predicate (relation) on \(D\), i.e., \(I(p) \subseteq D^n\).
    \[ I = [0 \leftrightarrow \text{zero}, + \leftrightarrow \text{add}, \ < \leftrightarrow \text{less-than}, \ldots] \]

- **Assignment** \(a : \text{Var} \rightarrow D\)
  - A function that maps every variable \(x\) to a value \(a(x)\) in this domain.
    \[ a = [x \leftrightarrow 1, y \leftrightarrow 0, z \leftrightarrow 3, \ldots] \]

The pair \(M = (D, I)\) is also called a **structure**.
The Semantics of Terms

\[ D, I, a \rightarrow \llbracket t \rrbracket \rightarrow d \in D \]

- Term semantics \( \llbracket t \rrbracket^D_I \in D \)
  - Given \( D, I, a \), the semantics of term \( t \) is a value in \( D \).
  - This value is defined by structural induction on \( t \).

\[ t ::= x \mid c \mid f(t_1, \ldots, t_n) \]

- \( \llbracket x \rrbracket^D_I := a(x) \)
  - The semantics of a variable is the value given by the assignment.

- \( \llbracket c \rrbracket^D_I := I(c) \)
  - The semantics of a constant is the value given by the interpretation.

- \( \llbracket f(t_1, \ldots, t_n) \rrbracket^D_I := I(f)(\llbracket t_1 \rrbracket^D_I, \ldots, \llbracket t_n \rrbracket^D_I) \)
  - The semantics of a function application is the result of the interpretation of the function symbol applied to the values of the argument terms.

The recursive definition of a function evaluating a term.
Example

\[ D = \mathbb{N} = \{\text{zero, one, two, three, \ldots}\} \]
\[ a = [x \mapsto \text{one}, y \mapsto \text{two}, \ldots] \]
\[ I = [0 \mapsto \text{zero}, + \mapsto \text{add}, \ldots] \]

\[
\left\lbrack x + (y + 0) \right\rbrack^{D,I}_a = \text{add}(\left\lbrack x \right\rbrack^{D,I}_a, \left\lbrack y + 0 \right\rbrack^{D,I}_a)
\]
\[
= \text{add}(a(x), \left\lbrack y + 0 \right\rbrack^{D,I}_a)
\]
\[
= \text{add}(\text{one}, \left\lbrack y + 0 \right\rbrack^{D,I}_a)
\]
\[
= \text{add}(\text{one}, \text{add}(\left\lbrack y \right\rbrack^{D,I}_a, \left\lbrack 0 \right\rbrack^{D,I}_a))
\]
\[
= \text{add}(\text{one}, \text{add}(a(y), I(0))
\]
\[
= \text{add}(\text{one}, \text{add}(\text{two}, \text{zero}))
\]
\[
= \text{add}(\text{one}, \text{two})
\]
\[
= \text{three}
\]

The meaning of the term with the “usual” interpretation.
Example

\[ D = \mathcal{P}(\mathbb{N}) = \{\emptyset, \{\text{zero}\}, \{\text{one}\}, \{\text{two}\}, \ldots, \{\text{zero}, \text{one}\}, \ldots\} \]
\[ a = [x \mapsto \{\text{one}\}, y \mapsto \{\text{two}\}, \ldots] \]
\[ I = [0 \mapsto \emptyset, + \mapsto \text{union}, \ldots] \]

\[
\llbracket x + (y + 0) \rrbracket_a^D, I = \text{union}(\llbracket x \rrbracket_a^D, I, \llbracket y + 0 \rrbracket_a^D, I)
\]
\[ = \text{union}(a(x), \llbracket y + 0 \rrbracket_a^D, I) \]
\[ = \text{union}(\{\text{one}\}, \llbracket y + 0 \rrbracket_a^D, I) \]
\[ = \text{union}(\{\text{one}\}, \text{union}(\llbracket y \rrbracket_a^D, I, \llbracket 0 \rrbracket_a^D, I)) \]
\[ = \text{union}(\{\text{one}\}, \text{union}(a(y), I(0))) \]
\[ = \text{union}(\{\text{one}\}, \text{union}(\{\text{two}\}, \text{emptyset})) \]
\[ = \text{union}(\{\text{one}\}, \{\text{two}\}) \]
\[ = \{\text{one}, \text{two}\} \]

The meaning of the term with another interpretation.
The Semantics of Formulas

\[ D, I, a \rightarrow [F] \rightarrow \text{true, false} \]

- Formula semantics \([ F ]_a^D, I \in \{ \text{true, false} \}

  ▶ Given \(D, I, a\), the semantics of term \(T\) is a truth value.
  ▶ This value is defined by structural induction on \(F\).

\[
F := p(t_1, \ldots, t_n) | \top | \bot \\
| \neg F | F_1 \land F_2 | F_1 \lor F_2 | F_1 \rightarrow F_2 | F_1 \leftrightarrow F_2 \\
| \forall x : F | \exists x : F | \ldots
\]

- \([ p(t_1, \ldots, t_n) ]_a^D, I := I(p)([t_1]_a^D, I, \ldots, [t_n]_a^D, I) \]
  ▶ The semantics of a atomic formula is the result of the interpretation of the predicate symbol applied to the values of the argument terms.

- \([ \top ]_a^D, I := \text{true}, [\bot]_a^D, I := \text{false}\)

And now for the non-atomic formulas.
The Semantics of Propositional Formulas

- \( \llbracket \neg F \rrbracket_{a}^{D,I} := \begin{cases} \text{true} & \text{if } \llbracket F \rrbracket_{a}^{D,I} = \text{false} \\ \text{false} & \text{else} \end{cases} \)

- \( \llbracket F_1 \land F_2 \rrbracket_{a}^{D,I} := \begin{cases} \text{true} & \text{if } \llbracket F_1 \rrbracket_{a}^{D,I} = \llbracket F_2 \rrbracket_{a}^{D,I} = \text{true} \\ \text{false} & \text{else} \end{cases} \)

- \( \llbracket F_1 \lor F_2 \rrbracket_{a}^{D,I} := \begin{cases} \text{false} & \text{if } \llbracket F_1 \rrbracket_{a}^{D,I} = \llbracket F_2 \rrbracket_{a}^{D,I} = \text{false} \\ \text{true} & \text{else} \end{cases} \)

- \( \llbracket F_1 \rightarrow F_2 \rrbracket_{a}^{D,I} := \begin{cases} \text{false} & \text{if } \llbracket F_1 \rrbracket_{a}^{D,I} = \text{true} \text{ and } \llbracket F_2 \rrbracket_{a}^{D,I} = \text{false} \\ \text{true} & \text{else} \end{cases} \)

- \( \llbracket F_1 \leftrightarrow F_2 \rrbracket_{a}^{D,I} := \begin{cases} \text{true} & \text{if } \llbracket F_1 \rrbracket_{a}^{D,I} = \llbracket F_2 \rrbracket_{a}^{D,I} \\ \text{false} & \text{else} \end{cases} \)

The semantics coincides here with that of propositional logic.
The Semantics of Quantified Formulas

\[ \left[ \forall x : F \right]^{D,I}_{a} := \begin{cases} \text{true} & \text{if } \left[ F \right]^{D,I}_{a[x \mapsto d]} = \text{true} \text{ for all } d \in D \\ \text{false} & \text{else} \end{cases} \]

- Formula is true, if body \( F \) is true for every value of the domain assigned to \( x \).

\[ \left[ \exists x : F \right]^{D,I}_{a} := \begin{cases} \text{true} & \text{if } \left[ F \right]^{D,I}_{a[x \mapsto d]} = \text{true} \text{ for some } d \in D \\ \text{false} & \text{else} \end{cases} \]

- Formula is true, if body \( F \) is true for at least one value of the domain assigned to \( x \).

\[ a[x \mapsto d](y) = \begin{cases} d & \text{if } x = y \\ a(y) & \text{else} \end{cases} \]
Example

\[ D = \mathbb{N}_3 = \{ \text{zero, one, two} \} \]

\[ a = [x \mapsto \text{one}, y \mapsto \text{two}, z \mapsto \text{two}, \ldots], \quad l = [0 \mapsto \text{zero}, + \mapsto \text{add}, \ldots] \]

\[ \forall x : \exists y : x + y = z \]_{a,l}^{D} = true

\[ \exists y : x + y = z \]_{a, \{x\mapsto\text{zero}\}}^{D, l} = true

\[ x + y = z \]_{a, \{x\mapsto\text{zero}, y\mapsto\text{zero}\}}^{D, l} = false

\[ x + y = z \]_{a, \{x\mapsto\text{zero}, y\mapsto\text{one}\}}^{D, l} = false

\[ x + y = z \]_{a, \{x\mapsto\text{zero}, y\mapsto\text{two}\}}^{D, l} = true

\[ \exists y : x + y = z \]_{a, \{x\mapsto\text{one}\}}^{D, l} = true

\[ x + y = z \]_{a, \{x\mapsto\text{one}, y\mapsto\text{zero}\}}^{D, l} = false

\[ x + y = z \]_{a, \{x\mapsto\text{one}, y\mapsto\text{one}\}}^{D, l} = true

\[ x + y = z \]_{a, \{x\mapsto\text{one}, y\mapsto\text{two}\}}^{D, l} = false

\[ \exists y : x + y = z \]_{a, \{x\mapsto\text{two}\}}^{D, l} = true

\[ x + y = z \]_{a, \{x\mapsto\text{two}, y\mapsto\text{zero}\}}^{D, l} = true

\[ x + y = z \]_{a, \{x\mapsto\text{two}, y\mapsto\text{one}\}}^{D, l} = false

\[ x + y = z \]_{a, \{x\mapsto\text{two}, y\mapsto\text{two}\}}^{D, l} = false

The systematic investigation of respectively search for assignments.
Semantic Notions

Let $F$ denote formulas, $M$ structures, $a$ assignments.

- $F$ is **satisfiable**, if $\llbracket F \rrbracket^M_a = true$ for some $M$ and $a$.
  
  $p(0, x)$ is satisfiable; $q(x) \land \neg q(x)$ is not.

- $M$ is a **model** of $F$ (short: $M \models F$), if $\llbracket F \rrbracket^M_a = true$ for all $a$.
  
  $(\mathbb{N}, [0 \mapsto \text{zero}, p \mapsto \text{less-equal}]) \models p(0, x)$

- $F$ is **valid** (short: $\models F$), if $M \models F$ for all $M$.
  
  $\models p(x) \land (p(x) \rightarrow q(x)) \rightarrow q(x)$

  - $F$ is satisfiable, if $\neg F$ is not valid.
  - $F$ is valid, if $\neg F$ is not satisfiable.

- $F$ is a **logical consequence** of formula set $\Gamma$ (short: $\Gamma \models F$), if for all $M$ and $a$, the following is true:
  
  If $\llbracket G \rrbracket^M_a = true$ for every $G$ in $\Gamma$, then also $\llbracket F \rrbracket^M_a = true$.

  $p(x), p(x) \rightarrow q(x) \models q(x)$

- $F_1$ is a **logical consequence** of formula $F_2$, if $\{ F_2 \} \models F_1$.  

Wolfgang Schreiner and Wolfgang Windsteiger

http://www.risc.jku.at
Logical Equivalence

We are now going to address the first question stated in the beginning.

- **Definition:** two formulas $F_1$ and $F_2$ are logically equivalent (short: $F_1 \equiv F_2$), if $F_1 \models F_2$ and $F_2 \models F_1$.

- **Lemma:** if $F \equiv F'$ and $G \equiv G'$, then

  $\neg F \equiv \neg F'$

  $F \land G \equiv F' \land G'$

  $F \lor G \equiv F' \lor G'$

  $F \rightarrow G \equiv F' \rightarrow G'$

  $F \leftrightarrow G \equiv F' \leftrightarrow G'$

  $\forall x : F \equiv \forall x : F'$

  $\exists x : F \equiv \exists x : F'$

Logically equivalent formulas can be substituted in any context without affecting the logical equivalence of the result (since $F \equiv G$ iff $F \equiv G$ is valid, this justifies the proof rule A-\(\leftrightarrow\)).
Expressiveness of First-Order Logic

- Variables denote elements of the domain, thus no quantification is possible over functions and predicates of the domain. 

  *This would require second-order predicate logic.*

- Nevertheless we express in first-order logic statements such as

  \[
  \forall A, B, f \in A \rightarrow B : f \text{ is bijective} \rightarrow \exists g \in B \rightarrow A : \forall x \in B : f(g(x)) = x
  \]

- This is possible because formulas are usually interpreted over the domain of sets, i.e., all variables denote sets:

  \[
  A \rightarrow B := \{ S \subseteq A \times B \mid \begin{align*}
  & (\forall a \in A : \exists b \in B : (a, b) \in S) \land \\
  & (\forall a, a', b : (a, b) \in S \land (a', b) \in S \rightarrow a = a')
  \end{align*}
  \}
  \]

- Terms like \( f(g(x)) \) involve a hidden binary function “apply”

  \[
  f(g(x)) \leadsto \text{apply}(f, \text{apply}(g, x))
  \]

  which denotes “function application”:

  \[
  \text{apply}(f, x) := \text{the } y : (x, y) \in f
  \]

First-order predicate logic over the domain of sets is the “working horse” of mathematics; virtually all of mathematics is formulated in this framework.
Now we turn our attention to the second question.

**Completeness Theorem (Kurt Gödel, 1929):** First order predicate logic has a proof calculus for which the following holds:

- **Soundness:** if by the rules of the calculus a conclusion $F$ can be derived from a set of assumptions $\Gamma$ ($\Gamma \vdash F$), then $F$ is a logical consequence of $\Gamma$ ($\Gamma \models F$).

- **Completeness:** if $F$ is a logical consequence of $\Gamma$ ($\Gamma \models F$), then by the rules of the calculus $F$ can be derived from $\Gamma$ ($\Gamma \vdash F$).

No logic that is stronger (more expressive) than first order predicate logic has a proof calculus that also enjoys both soundness and completeness.
Undecidability of First-Order Logic

The existence of a complete proof calculus does not mean that the truth of every formula is algorithmically decidable.

- **Undecidability (Church/Turing, 1936/1937):** there does not exist any algorithm that for given formula set \( \Gamma \) and formula \( F \) always terminates and says whether \( \Gamma \models F \) holds or not.

- **Semidecidability:** but there exists an algorithm, that for given \( \Gamma \) and \( F \), if \( \Gamma \models F \), detects this fact in a finite amount of time.

  This algorithm searches for a proof of \( \Gamma \vdash F \) in a complete proof calculus; if such a proof exists, it will eventually detect it; however, if no such proof exists, the search runs forever.

Automatic proof search is not able to detect that a formula is not true.
Limits of First-Order Logic

Not every structure can be completely described by a finite set of formulas.

- **Incompleteness Theorem (Kurt Gödel, 1931):** it is in no sound logic possible to prove all true arithmetic statements (i.e., all statements about natural numbers with addition and multiplication).
  - To adequately characterize $\mathbb{N}$, the (infinite) axiom scheme of mathematical induction has to be added.
- **Corollary:** in every sound formal system that is sufficiently rich there are statements that can neither be proved nor disproved.

In practice, complete reasoners for first-order logic are often supported by (complete or incomplete) reasoners for special theories.