First Order Predicate Logic

Syntax and Informal Semantics

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

Why Predicate Logic?

- Propositional logic is about "sentences" and their combination. "Sentence" $\rightsquigarrow$ something that can be true or false.
- Propositional logic cannot describe:
  1. "concrete objects" of a certain domain,
  2. functional relationships,
  3. statements about "for all" objects or about "for some" objects.
- (First order) Predicate logic is an extension of propositional logic, which (among other things!) allows to express these.

Why Predicate Logic?

ad 1. "Tom likes Alex":

- Propositional logic: $TLA \rightsquigarrow$ no reference to Tom and Alex.
- Predicate logic: $\text{likes}(\text{Tom}, \text{Alex})$.

ad 2. "Tom likes his brother":

- Propositional logic: $TLB \rightsquigarrow$ no relation between Tom and his brother.
- Predicate logic: $\text{likes}(\text{Tom}, \text{B}(\text{Tom}))$.

ad 3. "Tom likes all his classmates":

- Propositional logic:

  $$TLC_1 \wedge TLC_2 \wedge \ldots \wedge TLC_n$$

  $\rightsquigarrow$ cumbersome, lacks info that Tom is part of every clause.
- Predicate logic: $\forall x : C(x, \text{Tom}) \rightarrow \text{likes}(\text{Tom}, x)$.

Why Predicate Logic?

ad 3. "Everybody in Tom's class has a friend in the class":

- Propositional logic:

$$(C_1 L C_2 \lor C_1 L C_3 \lor \ldots \lor C_1 L C_n) \land (C_2 L C_1 \lor C_2 L C_3 \lor \ldots \lor C_2 L C_n) \land \\ \land \ldots \land (C_n L C_1 \lor C_n L C_2 \lor \ldots \lor C_n L C_{n-1})$$

  $\rightsquigarrow$ Extremely cumbersome, lacks structural info.
- Predicate logic: $\forall x : C(x, \text{Tom}) \rightarrow \exists y : C(y, \text{Tom}) \land \text{likes}(x, y)$.

ad 3. "Any natural number greater than 1 is a product of prime numbers":

- Propositional logic: $PP_2 \land PP_3 \land \ldots$ $\rightsquigarrow$ What do the ... mean?
- Predicate logic: $\forall n > 1 : \text{PP}(n)$.

ad 3. "Every positive real number has a square root":

- Propositional logic: $\rightsquigarrow$???
- Predicate logic: $\forall x : \text{PR}(x) \rightarrow \text{hSQ}(x)$.

Example

"All prime numbers greater than 2 are odd."

Formulation in propositional logic:

$$P \wedge G \rightarrow O, \quad \text{where} \begin{cases} P \ \ldots \text{a number is prime} \\ G \ \ldots \text{a number is greater than 2} \\ O \ \ldots \text{a number is odd} \end{cases}$$

Is this true? Check truth table $\rightsquigarrow$ sometimes true, sometimes false.

But: We are talking about concrete propositions about numbers, about primality, ordering, divisibility, etc. There are interrelations between these concepts, which are lost when transforming to abstract propositions P, G, and O. Propositional logic cannot be used to express these interrelations!

Example

"All prime numbers greater than 2 are odd."

Predicate logic has variables and quantifiers.

$P(x)\ldots$ x is a prime number	$\forall y : y \mid x \rightarrow y = 1 \vee y = x$
$G(x)\ldots$	$x > 2$
$O(x)\ldots$ x is odd	$\neg(2 \mid x)$

Formulation in predicate logic:

$$\forall x : (P(x) \wedge G(x)) \rightarrow O(x)$$

The interrelations between $P(x)$ and $O(x)$ (via divisibility) are preserved!

We will study techniques, how we can justify that this statement is always true.

Abstract Syntax vs. Concrete Syntax

Syntax = the form of expressions in the language.

Semantics = the meaning of expressions in the language.

- Abstract syntax: one particular standard form to describe expressions.
- Concrete syntax: "concrete way" to write/display expressions.
- Notation: just another word for concrete syntax.

Abstract syntax must allow unique identification of "type of the expression" and its "subexpressions".

One expression in abstract syntax can have many different forms in concrete syntax.

The language of mathematics is very rich in notations (e.g. subscripts, superscripts, writing things one above the other, etc.).

Well-chosen notation should convey intuitive meaning.

Syntax: Examples

a is less than b

- Abstract syntax: $<(a, b)$
- Notation: $a < b$

The open interval between a and b

- Abstract syntax: $\text{openInterval}(a, b)$
- Notation: $]a, b[$, (a, b)

The remainder of a divided by b

- Abstract syntax: $\text{remainder}(a, b)$
- Notation: $\text{mod}(a, b)$, $a \bmod p$, $a \pmod{p}$, $a\%b$

f converges to a

- Abstract syntax: $\text{converges}(f, a)$
- Notation: $f \to a$, $\lim f = a$, $f(n) \xrightarrow{n\to\infty} a$, $\lim\limits_{n\to\infty} f(n) = a$

Syntax: Terms and Formulae

In mathematics we want to speak about objects and properties of these objects.

The language of predicate logic provides terms and formulas.

Constituents of the language:

- variables, e.g. x, y, n, ...
- constants, e.g. 0, 1, π, e, ...
- function symbols, e.g. f, openInterval, remainder, ...
- predicate symbols, e.g. $=$, converges, $\leq$, ...
- logical connectives, e.g. $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, ...
- quantifiers, e.g. $\forall$, $\exists$, ...

Every function/predicate symbol has an *arity* (a natural number) associated.

Constants, function/predicate symbols and variables are names (of a certain form). We assume that every symbol belongs to exactly one of these categories.

Syntax: Terms

Terms should stand for objects.

- Every constant is a term.
- Every variable is a term.
- If f is a function symbol with arity $n \geq 1$ and $t_1, \ldots, t_n$ are terms, then

$$f(t_1, \ldots, t_n) \quad \text{is a term.}$$

A term is an expression that can be generated by finitely many applications of the above rules.

Syntax: Terms

In BNF-like form:

$$\mathit{term} ::= \mathit{constant} \mid \mathit{variable} \mid \mathit{fun\_sym} \text{ `(' } \mathit{term} \;\; (\text{`,' } \mathit{term})* \text{ `)'}$$

Example

Let s be a function symbol with arity 1 and y a variable. Then $s(y)$ is a term.

Let ‘remainder’ be a function symbol with arity 2, a and b constants. Then remainder(a, b) and remainder$(a, s(a))$ are terms.

Let ‘openInterval’ be a function symbol with arity 2, a and b constants. Then openInterval(a, b) is a term.

Syntax: Formulas

Formulas should stand for properties that can be true or false.

- $\top$ and $\bot$ are formulas.
- If p is a predicate symbol with arity $n \geq 1$ and $t_1, \ldots, t_n$ are terms, then

$$p(t_1, \ldots, t_n) \quad \text{is a formula.}$$

- If F is a formula, then $\neg F$ is a formula.
- If F_1 and F_2 are formulas, then the following are formulas:

$$F_1 \wedge F_2 \qquad F_1 \vee F_2 \qquad F_1 \rightarrow F_2 \qquad F_1 \leftrightarrow F_2$$

- If F is a formula and x a variable, then the following are formulas:

$$\forall x : F \qquad \exists x : F$$

A formula is an expression that can be generated by finitely many applications of the above rules.

Syntax: Formulas

In BNF-like form:

$$
\begin{aligned}
\textit{formula} &::= \top \mid \bot \mid \textit{atomic\_f} \mid \textit{connective\_f} \mid \textit{quantifier\_f} \\
\textit{atomic\_f} &::= \textit{pred\_sym} \text{ `('}\ \textit{term}\ \ (\text{`,'}\ \textit{term}\ )*\ \text{`)'} \\
\textit{connective\_f} &::= \textit{conn1}\ \ \textit{formula} \mid \textit{formula}\ \ \textit{conn2}\ \ \textit{formula} \\
\textit{conn1} &::= \neg \\
\textit{conn2} &::= \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow \\
\textit{quantifier\_f} &::= \textit{quantifier}\ \ \textit{variable} \text{ `:'}\ \textit{formula} \\
\textit{quantifier} &::= \forall \mid \exists
\end{aligned}
$$

Syntax: Formulas

Example

Let '$=$' be a predicate symbol with arity 2, a and b constants. Then $=(a,b)$ and also $\neg(=(a,b))$ are formulas.

Let $<$ be predicate symbol with arity 2. Then $\text{remainder}(a,b) < b$ is a formula.

Let 'converges' be a predicate symbol with arity 2, f and a variables, 0 a constant. Then the following are formulas:

$$\text{converges}(f,a) \vee = (a,0)$$

$$\forall f : \text{converges}(f,a) \wedge = (a,0)$$

$$\forall f : \forall a : \text{converges}(f,a) \leftrightarrow = (a,0)$$

$$\forall f : \text{converges}(f,a) \rightarrow \exists a : = (a,0)$$

Syntax: Notations and Conventions

- Function/Predicate symbols are often written using infix/prefix/postfix/matchfix operators:

$$a < b \rightsquigarrow < (a, b) \qquad \int f \rightsquigarrow \int (f)$$

$$\frac{a}{b} \rightsquigarrow /(a, b) \qquad]a, b[\rightsquigarrow \mathrm{openInterval}(a, b)$$

$$f' \rightsquigarrow \mathrm{derivative}(f) \qquad f \to a \rightsquigarrow \mathrm{converges}(f, a)$$

- Variable arity (overloading, no details):

$$a + b \rightsquigarrow +(a, b) \qquad a + b + c \rightsquigarrow \begin{cases} +(a, b, c) \\ +(+(a, b), c) \\ +(a, +(b, c)) \end{cases}$$

(beyond syntax!)

Syntax: Free and Bound Variables

Every occurrence of x in $\forall x : F$ is called bound (by the $\forall$-quantifier).

Every occurrence of x in $\exists x : F$ is called bound (by the $\exists$-quantifier).

An occurence of a variable is called free if it is not bound.

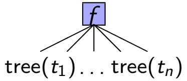
Example

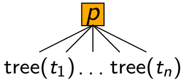
$$\begin{array}{rcl}
\text{converges}(f, a) \vee a = 0 & \rightsquigarrow & \text{no bound vars., } f, a \text{ free} \\
\forall f : \text{converges}(f, a) \wedge a = 0 & \rightsquigarrow & f \text{ is bound, } a \text{ is free} \\
\forall f : \forall a : \text{converges}(f, a) \leftrightarrow a = 0 & \rightsquigarrow & f, a \text{ are bound, no free vars.} \\
\forall f : \text{converges}(f, a) \rightarrow \exists a : = (a, 0) & \rightsquigarrow & f \text{ bound, } a \text{ free and bound.}
\end{array}$$

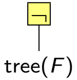
Syntax Analysis

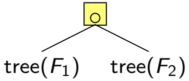
- Constants, variables, function symbols, and predicate symbols should be distinguishable.
- Function/Predicate symbols are often not specified explicitly but must be recognized in mathematical expressions $\rightsquigarrow$ syntax analysis.
- Reveal the exact syntactical structure of an expression $\rightsquigarrow$ syntax tree.
- Determine, which variables are free/bound.

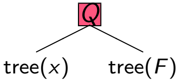
Syntax Analysis

- Constant c: $\text{tree}(c) = \boxed{c}$
- Variable x: $\text{tree}(x) = \boxed{x}$
- Term $f(t_1, \ldots, t_n)$: $\text{tree}(f(t_1, \ldots, t_n)) =$ 

- Formula $p(t_1, \ldots, t_n)$: $\text{tree}(p(t_1, \ldots, t_n)) =$ 

- Formula $\neg F$: $\text{tree}(\neg F) =$ 

- For $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$: $\text{tree}(F_1 \circ F_2) =$ 

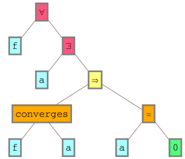
- For $Q \in \{\forall, \exists\}$: $\text{tree}(Qx : F) =$ 

Syntax Analysis

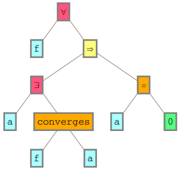
$\forall f : \exists a : \text{converges}(f, a) \rightarrow a = 0$ can be meant as any of

1. $\forall f : \exists a : (\text{converges}(f, a) \rightarrow a = 0)$
2. $\forall f : (\exists a : \text{converges}(f, a)) \rightarrow a = 0$
3. $(\forall f : \exists a : \text{converges}(f, a)) \rightarrow a = 0$

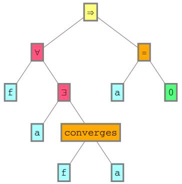
1.



2.



3.

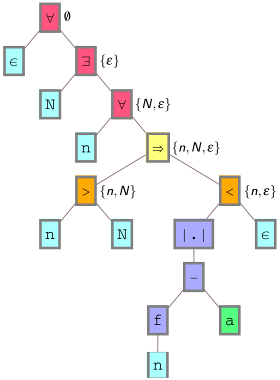


🟩	🟦	🟪	🟧	🟨	🟥
constant	variable	function symbol	predicate symbol	connective	quantifier

Syntax Analysis

$$\forall \varepsilon : \exists N : \forall n : (n > N \rightarrow |f(n) - a| < \varepsilon)$$

- Quantifiers ✓
- Left and right of $\rightarrow$ must be formulas.
- $n > N$ must be an atomic formula (infix notation, predicate symbol "$>$" applied to variables n and N.
- $|f(n) - a| < \varepsilon$: predicate symbol "$<$" applied to $|f(n) - a|$ and the variable ε.
- $|f(n) - a|$: function symbol "$|.|$" applied to $f(n) - a$.
- $f(n) - a$: function symbol "$-$" applied to $f(n)$ and a.
- $f(n)$: function symbol f applied to variable n.



Semantics

A predicate logic expression gets a meaning through a configuration, i.e. the specification of

1. a non-empty domain,
2. an interpretation that gives
   - for every constant an element of that domain,
   - for every function symbol with arity n some concrete n-ary function on the domain, and
   - for every predicate symbol with arity n some concrete n-ary relation on the domain, and
3. an assignment for the free variables in the expression.

Semantics of Terms and Formulas

Meaning of a term is an object in the domain.

- Meaning of a variable $\rightsquigarrow$ assignment.
- Meaning of a constant $\rightsquigarrow$ interpretation.
- Meaning of $f(t_1, \ldots, t_n) \rightsquigarrow$ apply the interpretation of f to the meaning of the t_i.

Meaning of a formula is true or false.

- Meaning of $\top$ is true, meaning of $\bot$ is false.
- Meaning of $t_1 = t_2 \rightsquigarrow$ the meanings of t_1 and t_2 are identical.
- Meaning of $p(t_1, \ldots, t_n) \rightsquigarrow$ apply the interpretation of p to the meaning of the t_i.
- Meaning of logical connectives $\rightsquigarrow$ apply truth tables to the meaning of the constituent subformulas.
- Meaning of $\forall x : F \rightsquigarrow$ true iff the meaning of F is true for all possible assignments for the free variable x.
- Meaning of $\exists x : F \rightsquigarrow$ true iff the meaning of F is true for at least one assignment for the free variable x.

Semantics: Examples

$\forall n : R(n, n)$

- Domain: natural numbers.
- R is interpreted as the divisibility relation on natural numbers.
- Every natural number is divisible by itself. $\rightsquigarrow$ true

$\forall n : R(n, n)$

- Domain: real numbers.
- R is interpreted as the less-than relation on real numbers.
- Every real number is less than itself. $\rightsquigarrow$ false

$\exists x : R(a, x) \wedge R(x, b)$

- Domain: real numbers.
- R is interpreted as the less-than relation on real numbers.
- There is a real number x such that $a < x$ and $x < b$. $\rightsquigarrow$???
- Assignment $[a \mapsto 5, b \mapsto 6]$: There is an assignment for x such that $5 < x$ and $x < 6$. $\rightsquigarrow$ true, e.g. $[x \mapsto 5.5]$
- Assignment $[a \mapsto 7, b \mapsto 6]$: There is an assignment for x such that $7 < x$ and $x < 6$. $\rightsquigarrow$ false, why?

Nested Quantifiers

When quantifiers of different type are nested, the order matters.

Example

Domain: natural numbers.

$$\forall x : \exists y : x < y \quad \rightsquigarrow \textit{true}$$

(Why? For the assignment $[x \mapsto \bar{x}]$ for x take $[y \mapsto \bar{x}+1]$ as the assignment for y. The meaning of $x < y$ is then $\bar{x} < \bar{x}+1$, which is true no matter what $\bar{x}$ is.)

$$\exists y : \forall x : x < y \quad \rightsquigarrow \textit{false}$$

(Why? Assume it was true, i.e. there is an assignment $[y \mapsto \bar{y}]$ for y such that $x < y$ is true for all assignments for x. But take $[x \mapsto \bar{y}]$ as the assignment for x. The meaning of $x < y$ is then $\bar{y} < \bar{y}$, which is false, hence the original assumption must not be made, thus the meaning of the formula must be false.)

Semantics Convention

Meaning of "$=$", logical connectives, and quantifiers $\rightsquigarrow$ defined by the above rules.

The meaning of all other symbols $\rightsquigarrow$ interpretation $\rightsquigarrow$ can be chosen as desired and must be given explicitly.

In principle possible: express "a divides the sum of b and c" by

$$a \subseteq (b * c)$$

using the interpretation

$$[\subseteq \mapsto \text{ the divisibility relation, } * \mapsto \text{ the addition function}].$$

Convention: interpretation is not given explicitly, a "standard interpretation" is assumed.

Semantics: Consequence and Equivalence

F is a (logical) consequence of Γ if
F is true in every configuration, in which all $G \in \Gamma$ are true.

- F_2 is a logical consequence of F_1 means F_2 is a consequence of $\{F_1\}$.
- F_2 "follows from" F_1 regardless of the configuration. F_1 "implies" F_2.

F_1 is (logically) equivalent to F_2 (write "$F_1 \Leftrightarrow F_2$") if
F_1 is a consequence of F_2 and F_2 is a consequence of F_1.

- F_1 and F_2 have the same meaning, regardless of the configuration.
- Every formula can always be substituted by an equivalent one.

F is valid if F is true in every configuration.

- F is a "fact", F is a logical consequence of $\emptyset$.
- $F_1 \Leftrightarrow F_2$ iff ($F_1 \leftrightarrow F_2$ is valid).
- F_2 is a logical consequence of F_1 iff ($F_1 \rightarrow F_2$ is valid).

Equivalent Formulas

In addition to equivalences for connectives (see propositional logic):

$$
\begin{array}{lcl}
\neg(\forall x : F) & \Leftrightarrow & \exists x : \neg F \quad \text{(De-Morgan)} \\
\neg(\exists x : F) & \Leftrightarrow & \forall x : \neg F \quad \text{(De-Morgan)} \\
\forall x : (F_1 \wedge F_2) & \Leftrightarrow & (\forall x : F_1) \wedge (\forall x : F_2) \\
\exists x : (F_1 \vee F_2) & \Leftrightarrow & (\exists x : F_1) \vee (\exists x : F_2) \\
\forall x : (F_1 \vee F_2) & \Leftrightarrow & F_1 \vee (\forall x : F_2), \text{ if } x \text{ does not occur free in } F_1 \\
\exists x : (F_1 \wedge F_2) & \Leftrightarrow & F_1 \wedge (\exists x : F_2), \text{ if } x \text{ does not occur free in } F_1
\end{array}
$$

For a finite domain $\{v_1, \ldots, v_n\}$:

$$
\begin{array}{lcl}
\forall x : F & \Leftrightarrow & F[v_1/x] \wedge \ldots \wedge F[v_n/x] \\
\exists x : F & \Leftrightarrow & F[v_1/x] \vee \ldots \vee F[v_n/x]
\end{array}
$$

$E[t/x]$: the expression E with every free occurrence of x substituted by the term t. ($\rightsquigarrow$ E has the same meaning for x as $E[t/x]$ has for t.)

Natural Language Formulations in Predicate Logic

- Alex is Tom's sister.

$$\text{sister}(\text{Alex}, \text{Tom})$$

- Tom has a sister in Linz.

$$\exists x : \text{sister}(x, \text{Tom}) \wedge \text{lives-in}(x, \text{Linz})$$

- Tom has two sisters.

$$\exists x, y : x \neq y \wedge \text{sister}(x, \text{Tom}) \wedge \text{sister}(y, \text{Tom})$$

- Tom has no brother.

$$\begin{array}{ll} \neg\exists x : \text{brother}(x, \text{Tom}) & \text{i.e. there does not exist a brother of Tom} \\ \forall x : \neg\text{brother}(x, \text{Tom}) & \text{i.e. everybody is not a brother of Tom} \end{array}$$

Conditions in Quantifiers

We want: quantifier does not range over entire domain, “filter” values by a condition C.

Solution:

$$\forall x : C \rightarrow F \qquad\qquad \exists x : C \wedge F$$

Notation:

$$\forall C : F \qquad\qquad \exists C : F$$

The quantified variable must be recognized from C.

Example

$$\forall x \in \mathbb{N} : x | 10 \qquad\qquad \exists x < 10 : x | 10$$

Language Extensions

1. Locally bound variables: **let** $x = t$ **in** E
   - E can be a term or a formula, **let** … **in** … is term or a formula, respectively.
   - Binds the variable x.
   - Meaning: $E[t/x]$.
   - Alternative notation: E **where** $x = t$ or $E|_{x=t}$.
   - If F is a formula, then

$$\textbf{let } x = t \textbf{ in } F \Leftrightarrow \exists x : x = t \wedge F.$$

2. Conditional: **if** C **then** E_1 **else** E_2
   - E_i can be both terms or both formulas, **if** C **then** E_1 **else** E_2 is term or a formula, respectively.
   - Meaning: if C means true, then the meaning of E_1, otherwise the meaning of E_2.
   - If E_1 and E_2 are formulas, then

$$\textbf{if } C \textbf{ then } E_1 \textbf{ else } E_2 \Leftrightarrow (C \rightarrow E_1) \wedge (\neg C \rightarrow E_2).$$

Further Quantifiers

Common mathematical language uses more quantifiers:

- $\sum_{i=l}^{h} t$: binds i. Meaning: $t[l/i] + \cdots + t[h/i]$.
- $\prod_{i=l}^{h} t$: binds i. Meaning: $t[l/i] \cdots t[h/i]$.
- $\{x \in A \mid P\}$: binds x. Meaning: The set of all x in A such that P is true.
- $\{t \mid x \in A \wedge P\}$: binds x. Meaning: The set of all t when x is in A and P is true.
- $\lim_{x \to v} t$: binds x. Meaning: The limit of t when x goes to v.
- $\max_{x \in A} t$: binds x. Meaning: The maximum of t when x runs through A.
- $\min_{x \in A} t$: binds x. Meaning: The minimum of t when x runs through A.
- ...