Logic Programming

The Theoretical Basis of Logic Programming

December 2, 2015

- We review here (first order) predicate logic:

- We review here (first order) predicate logic:
  - the syntax,

- We review here (first order) predicate logic:
  - the syntax,
  - the semantics,

- We review here (first order) predicate logic:
  - the syntax,
  - the semantics,
  - illustrate some difficulties of the semantic evaluation of truth in first order logic,

- We review here (first order) predicate logic:
  - the syntax,
  - the semantics,
  - illustrate some difficulties of the semantic evaluation of truth in first order logic,
  - review some results that deal with this difficulty.

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - ( ),

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - ( ),
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - $(\;)$,
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - ( ),
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.
  - The set of variables $\mathcal{V}$ (countable set).

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - $($ $)$,
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.
  - The set of variables $\mathcal{V}$ (countable set).
  - The set of language symbols $\mathcal{L}$:

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - ( ),
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.
  - The set of variables $\mathcal{V}$ (countable set).
  - The set of language symbols $\mathcal{L}$:
    - $\mathcal{F}$ - function symbols (each with their own arity),

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - $($ $)$,
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.
  - The set of variables $\mathcal{V}$ (countable set).
  - The set of language symbols $\mathcal{L}$:
    - $\mathcal{F}$ - function symbols (each with their own arity),
    - $\mathcal{P}$ - predicate symbols (with arity),

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - ( ),
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.
  - The set of variables $\mathcal{V}$ (countable set).
  - The set of language symbols $\mathcal{L}$:
    - $\mathcal{F}$ - function symbols (each with their own arity),
    - $\mathcal{P}$ - predicate symbols (with arity),
    - $\mathcal{C}$ - constant symbols.

Syntax of first order predicate logic

- The **vocabulary** of the language contains the symbols from which expressions of the language are built:
  - Reserved symbols:
    - ( ),
    - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$,
    - $\forall, \exists$.
  - The set of variables $\mathcal{V}$ (countable set).
  - The set of language symbols $\mathcal{L}$:
    - $\mathcal{F}$ - function symbols (each with their own arity),
    - $\mathcal{P}$ - predicate symbols (with arity),
    - $\mathcal{C}$ - constant symbols.
- Example language (symbols):
  $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$. We use a notation similar to Prolog to indicate the arity of symbols.

- The expressions of (first order) predicate logic:

- The expressions of (first order) predicate logic:
  - Terms:

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,
    - constants $c \in \mathcal{C}$ are terms,

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,
    - constants $c \in \mathcal{C}$ are terms,
    - if $f_{/n} \in \mathcal{F}$ and $t_1, \ldots, t_n$ are terms, then so is $f(t_1, \ldots, t_n)$.

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,
    - constants $c \in \mathcal{C}$ are terms,
    - if $f_{/n} \in \mathcal{F}$ and $t_1, \ldots, t_n$ are terms, then so is $f(t_1, \ldots, t_n)$.
  - Formulae:

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,
    - constants $c \in \mathcal{C}$ are terms,
    - if $f_{/n} \in \mathcal{F}$ and $t_1, \ldots, t_n$ are terms, then so is $f(t_1, \ldots, t_n)$.
  - Formulae:
    - if $p_{/n} \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is an atomic formula,

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,
    - constants $c \in \mathcal{C}$ are terms,
    - if $f_{/n} \in \mathcal{F}$ and $t_1, \ldots, t_n$ are terms, then so is $f(t_1, \ldots, t_n)$.
  - Formulae:
    - if $p_{/n} \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is an atomic formula,
    - if F, G are formulae, then $\neg F$, $F \wedge G$, $F \vee G$, $F \Rightarrow G$, $F \Leftrightarrow G$ are (compound) formulae,

- The expressions of (first order) predicate logic:
  - Terms:
    - variables $v \in \mathcal{V}$ are terms,
    - constants $c \in \mathcal{C}$ are terms,
    - if $f_{/n} \in \mathcal{F}$ and $t_1, \ldots, t_n$ are terms, then so is $f(t_1, \ldots, t_n)$.
  - Formulae:
    - if $p_{/n} \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is an atomic formula,
    - if F, G are formulae, then $\neg F$, $F \wedge G$, $F \vee G$, $F \Rightarrow G$, $F \Leftrightarrow G$ are (compound) formulae,
    - if $x \in \mathcal{V}$ and F is a formula then $\forall x F$, $\exists x F$ are (quantified) formulae (the universally and existentially quantified formulae, respectively).

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,
  - **relations** between the objects (or properties of the objects),

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,
  - **relations** between the objects (or properties of the objects),
  - **processes** or **functions** that produce new objects from other objects.

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,
  - **relations** between the objects (or properties of the objects),
  - **processes** or **functions** that produce new objects from other objects.
- To find (compute) the meaning of an expression, one must first define an **interpretation** of the symbols:

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,
  - **relations** between the objects (or properties of the objects),
  - **processes** or **functions** that produce new objects from other objects.
- To find (compute) the meaning of an expression, one must first define an **interpretation** of the symbols:
  - constants are interpreted as objects in the domain described by the language,

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,
  - **relations** between the objects (or properties of the objects),
  - **processes** or **functions** that produce new objects from other objects.
- To find (compute) the meaning of an expression, one must first define an **interpretation** of the symbols:
  - constants are interpreted as objects in the domain described by the language,
  - function symbols are interpreted as processes (functions) in the domain described by the language,

Semantics of first order predicate logic

- The semantics of first order logic describes the meaning of expressions in the language.
- Such a language is used to describe:
  - a **domain** of objects,
  - **relations** between the objects (or properties of the objects),
  - **processes** or **functions** that produce new objects from other objects.
- To find (compute) the meaning of an expression, one must first define an **interpretation** of the symbols:
  - constants are interpreted as objects in the domain described by the language,
  - function symbols are interpreted as processes (functions) in the domain described by the language,
  - predicate symbols are interpreted as relations/properties between/of objects in the domain described by the language.

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",
    - $\mathcal{I}_2(1) =$ "one",

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",
    - $\mathcal{I}_2(1) =$ "one",
    - $\mathcal{I}_2(+) =$ concatenation,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",
    - $\mathcal{I}_2(1) =$ "one",
    - $\mathcal{I}_2(+) =$ concatenation,
    - $\mathcal{I}_2(-) =$ reverse,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",
    - $\mathcal{I}_2(1) =$ "one",
    - $\mathcal{I}_2(+) =$ concatenation,
    - $\mathcal{I}_2(-) =$ reverse,
    - $\mathcal{I}_2(<) =$ substring,

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",
    - $\mathcal{I}_2(1) =$ "one",
    - $\mathcal{I}_2(+) =$ concatenation,
    - $\mathcal{I}_2(-) =$ reverse,
    - $\mathcal{I}_2(<) =$ substring,
    - $\mathcal{I}_1(\geq) =$ sorted version.

- Consider the language presented previously $\mathcal{L} = \{\{+_{/2}, -_{/1}\}, \{<_{/2}, \geq_{/2}\}, \{0, 1\}\}$ and let's consider two interpretations of this language:
  - $\mathcal{I}_1$ an interpretation in the natural numbers:
    - $\mathcal{I}_1(0) =$ seven,
    - $\mathcal{I}_1(1) =$ zero,
    - $\mathcal{I}_1(+) =$ multiplication,
    - $\mathcal{I}_1(-) =$ factorial,
    - $\mathcal{I}_1(<) =$ smaller than,
    - $\mathcal{I}_1(\geq) =$ divides.
  - $\mathcal{I}_2$ an interpretation in domain of strings:
    - $\mathcal{I}_2(0) =$ " ",
    - $\mathcal{I}_2(1) =$ "one",
    - $\mathcal{I}_2(+) =$ concatenation,
    - $\mathcal{I}_2(-) =$ reverse,
    - $\mathcal{I}_2(<) =$ substring,
    - $\mathcal{I}_1(\geq) =$ sorted version.
- Note that interpretation shows the correspondence between the name of a concept (constant, function symbol, predicate symbol) and the concept described by that name.

- Once an interpretation has been defined, one can compute **the value of an expression E under interpretation $\mathcal{I}$,** $v_{\mathcal{I}}(E)$ (i.e. the meaning of an expression under interpretation) in the following way:

- Once an interpretation has been defined, one can compute **the value of an expression E under interpretation $\mathcal{I}$,** $\upsilon_{\mathcal{I}}(E)$ (i.e. the meaning of an expression under interpretation) in the following way:
- **The value of terms under interpretation**:

- Once an interpretation has been defined, one can compute **the value of an expression E under interpretation $\mathcal{I}$,** $v_{\mathcal{I}}(E)$ (i.e. the meaning of an expression under interpretation) in the following way:
- **The value of terms under interpretation**:
  - In general, terms will evaluate to objects in the universe of discourse.

- Once an interpretation has been defined, one can compute **the value of an expression E under interpretation $\mathcal{I}$,** $v_{\mathcal{I}}(E)$ (i.e. the meaning of an expression under interpretation) in the following way:
- **The value of terms under interpretation**:
  - In general, terms will evaluate to objects in the universe of discourse.
  - If $c \in \mathcal{C}$, $v_{\mathcal{I}}(c) = \mathcal{I}(c)$.

- Once an interpretation has been defined, one can compute **the value of an expression E under interpretation $\mathcal{I}$,** $\upsilon_{\mathcal{I}}(E)$ (i.e. the meaning of an expression under interpretation) in the following way:
- **The value of terms under interpretation**:
  - In general, terms will evaluate to objects in the universe of discourse.
  - If $c \in \mathcal{C}$, $\upsilon_{\mathcal{I}}(c) = \mathcal{I}(c)$.
  - If $x \in \mathcal{V}$, $\upsilon_{\mathcal{I}}(v)$ is not defined, unless the variable v is assigned a value. I.e. the value of expressions containing free variables cannot be determined unless the variables have values assigned to them.

- Once an interpretation has been defined, one can compute **the value of an expression E under interpretation $\mathcal{I}$,** $\upsilon_{\mathcal{I}}(E)$ (i.e. the meaning of an expression under interpretation) in the following way:
- **The value of terms under interpretation**:
  - In general, terms will evaluate to objects in the universe of discourse.
  - If $c \in \mathcal{C}$, $\upsilon_{\mathcal{I}}(c) = \mathcal{I}(c)$.
  - If $x \in \mathcal{V}$, $\upsilon_{\mathcal{I}}(v)$ is not defined, unless the variable v is assigned a value. I.e. the value of expressions containing free variables cannot be determined unless the variables have values assigned to them.
  - If $f(t_1, \ldots, t_n)$ is a term, then

$$\upsilon_{\mathcal{I}}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

- **The value of formulae under interpretation**:

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \wedge G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **true**.

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$v_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(v_{\mathcal{I}}(t_1), \ldots, v_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $v_{\mathcal{I}}(\neg F) =$ **true** iff $v_{\mathcal{I}}(F) =$ **false**.
    - $v_{\mathcal{I}}(F \wedge G) =$ **true** iff $v_{\mathcal{I}}(F) =$ **true** and $v_{\mathcal{I}}(G) =$ **true**.

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \wedge G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **true**.
    - $\upsilon_{\mathcal{I}}(F \vee G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** or $\upsilon_{\mathcal{I}}(G) =$ **true** (at least one is true).

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$v_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(v_{\mathcal{I}}(t_1), \ldots, v_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $v_{\mathcal{I}}(\neg F) =$ **true** iff $v_{\mathcal{I}}(F) =$ **false**.
    - $v_{\mathcal{I}}(F \wedge G) =$ **true** iff $v_{\mathcal{I}}(F) =$ **true** and $v_{\mathcal{I}}(G) =$ **true**.
    - $v_{\mathcal{I}}(F \vee G) =$ **true** iff $v_{\mathcal{I}}(F) =$ **true** or $v_{\mathcal{I}}(G) =$ **true** (at least one is true).
    - $v_{\mathcal{I}}(F \Rightarrow G) =$ **false** iff $v_{\mathcal{I}}(F) =$ **true** and $v_{\mathcal{I}}(G) =$ **false**.

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \wedge G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **true**.
    - $\upsilon_{\mathcal{I}}(F \vee G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** or $\upsilon_{\mathcal{I}}(G) =$ **true** (at least one is true).
    - $\upsilon_{\mathcal{I}}(F \Rightarrow G) =$ **false** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \Leftrightarrow G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) = \upsilon_{\mathcal{I}}(G)$.

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \wedge G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **true**.
    - $\upsilon_{\mathcal{I}}(F \vee G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** or $\upsilon_{\mathcal{I}}(G) =$ **true** (at least one is true).
    - $\upsilon_{\mathcal{I}}(F \Rightarrow G) =$ **false** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \Leftrightarrow G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) = \upsilon_{\mathcal{I}}(G)$.
  - For quantified formulae:

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \wedge G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **true**.
    - $\upsilon_{\mathcal{I}}(F \vee G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** or $\upsilon_{\mathcal{I}}(G) =$ **true** (at least one is true).
    - $\upsilon_{\mathcal{I}}(F \Rightarrow G) =$ **false** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \Leftrightarrow G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) = \upsilon_{\mathcal{I}}(G)$.
  - For quantified formulae:
    - $\upsilon_{\mathcal{I}}(\forall x F) =$ **true** iff for all values of x from the domain, $\upsilon_{\mathcal{I}}(F) =$ **true**.

- **The value of formulae under interpretation**:
  - Formulae will evaluate to **true** or **false** (but not both).
  - For atomic formulae,

$$\upsilon_{\mathcal{I}}(p(t_1, \ldots, t_n)) = \mathcal{I}(p)(\upsilon_{\mathcal{I}}(t_1), \ldots, \upsilon_{\mathcal{I}}(t_n)).$$

  - For compound formulae:
    - $\upsilon_{\mathcal{I}}(\neg F) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \wedge G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **true**.
    - $\upsilon_{\mathcal{I}}(F \vee G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) =$ **true** or $\upsilon_{\mathcal{I}}(G) =$ **true** (at least one is true).
    - $\upsilon_{\mathcal{I}}(F \Rightarrow G) =$ **false** iff $\upsilon_{\mathcal{I}}(F) =$ **true** and $\upsilon_{\mathcal{I}}(G) =$ **false**.
    - $\upsilon_{\mathcal{I}}(F \Leftrightarrow G) =$ **true** iff $\upsilon_{\mathcal{I}}(F) = \upsilon_{\mathcal{I}}(G)$.
  - For quantified formulae:
    - $\upsilon_{\mathcal{I}}(\forall x F) =$ **true** iff for all values of x from the domain, $\upsilon_{\mathcal{I}}(F) =$ **true**.
    - $\upsilon_{\mathcal{I}}(\exists x F) =$ **true** iff for some values of x from the domain, $\upsilon_{\mathcal{I}}(F) =$ **true**.

- For example, consider $\mathcal{I}_1$ as defined above:

$$
\begin{aligned}
& \upsilon_{\mathcal{I}_1}(-(0+1))) = \\
& \qquad \mathcal{I}_1(-)(\upsilon_{\mathcal{I}_1}(0+1)) = \\
& \qquad \mathsf{factorial}(\mathcal{I}_1(+)(\upsilon_{\mathcal{I}_1}(0), \upsilon_{\mathcal{I}_1}(1))) = \\
& \qquad \mathsf{factorial}(\mathsf{multiplication}(\mathsf{seven}, \mathsf{zero})) = \\
& \qquad \mathsf{factorial}(\mathsf{zero}) = \\
& \qquad \mathsf{one}.
\end{aligned}
$$

Validity, satisfiability, unsatisfiability

- We are interested in the meaning of formulae, in particular:

Validity, satisfiability, unsatisfiability

- We are interested in the meaning of formulae, in particular:
  - Whether a formula is **valid**, i.e. true under all possible interpretations.

Validity, satisfiability, unsatisfiability

- We are interested in the meaning of formulae, in particular:
  - Whether a formula is **valid**, i.e. true under all possible interpretations.
  - Whether a formula is **satisfiable**, i.e. there is an intepretation such that the formula is true.

Validity, satisfiability, unsatisfiability

- We are interested in the meaning of formulae, in particular:
  - Whether a formula is **valid**, i.e. true under all possible interpretations.
  - Whether a formula is **satisfiable**, i.e. there is an intepretation such that the formula is true.
  - Whether a formula is **unsatisfiable**, i.e. the formula is false under all possible interpretations.

Validity, satisfiability, unsatisfiability

- We are interested in the meaning of formulae, in particular:
  - Whether a formula is **valid**, i.e. true under all possible interpretations.
  - Whether a formula is **satisfiable**, i.e. there is an intepretation such that the formula is true.
  - Whether a formula is **unsatisfiable**, i.e. the formula is false under all possible interpretations.
  - Whether two formulae are **logically equivalent**, i.e. the formulae have the same meaning under all possible interpretations (we denote $F_1 \equiv F_2$).

Validity, satisfiability, unsatisfiability

- We are interested in the meaning of formulae, in particular:
  - Whether a formula is **valid**, i.e. true under all possible interpretations.
  - Whether a formula is **satisfiable**, i.e. there is an intepretation such that the formula is true.
  - Whether a formula is **unsatisfiable**, i.e. the formula is false under all possible interpretations.
  - Whether two formulae are **logically equivalent**, i.e. the formulae have the same meaning under all possible interpretations (we denote $F_1 \equiv F_2$).
  - Whether a formula is a **logical consequence** of a set of other formulae, i.e. the formula is true in all intepretations such that all formulae in the set are true (we denote $F_1, \ldots, F_n \models G$).

- Using these notions in practice is very difficult: the number of possible intepretations for a language is infinite. Checking the value of an expression in all possible intepretations is therefore not practical.

- Using these notions in practice is very difficult: the number of possible intepretations for a language is infinite. Checking the value of an expression in all possible intepretations is therefore not practical.
- If a formula is (or a set of formulae are) true under an interpretation in a domain, then that domain is called a **model** of the formula(e).

Herbrand Universe

- Fortunately, the difficulty represented by the immense number of possible interpretations of a language can be overcome.

Herbrand Universe

- Fortunately, the difficulty represented by the immense number of possible interpretations of a language can be overcome.
- We will define a domain and an interpretation that "captures" all the properties of all potential domains and interpretation.

Herbrand Universe

- Fortunately, the difficulty represented by the immense number of possible interpretations of a language can be overcome.
- We will define a domain and an interpretation that "captures" all the properties of all potential domains and interpretation.
- Checking satisfiability (and validity) of a formula (set) can be done by just checking the evaluation under a certain interpretation into this special universe.

Herbrand Universe

- Fortunately, the difficulty represented by the immense number of possible interpretations of a language can be overcome.
- We will define a domain and an interpretation that "captures" all the properties of all potential domains and interpretation.
- Checking satisfiability (and validity) of a formula (set) can be done by just checking the evaluation under a certain interpretation into this special universe.
- Let $\mathcal{L}$ be a language containing the constant symbols $\mathcal{C}$, function symbols $\mathcal{F}$ and predicate symbols $\mathcal{P}$. Let F be a formula over $\mathcal{L}$.

- The **Herbrand universe** $\mathcal{H}$ corresponding to the language $\mathcal{L}$ (or corresponding to the formula F) is defined in the following way:

- The **Herbrand universe** $\mathcal{H}$ corresponding to the language $\mathcal{L}$ (or corresponding to the formula F) is defined in the following way:
  - If $c \in \mathcal{C}$ then $c \in \mathcal{H}$.

- The **Herbrand universe** $\mathcal{H}$ corresponding to the language $\mathcal{L}$ (or corresponding to the formula F) is defined in the following way:
  - If $c \in \mathcal{C}$ then $c \in \mathcal{H}$.
  - If $t_1, \ldots, t_n$ and $f \in \mathcal{F}$ then $f(t_1, \ldots, t_n) \in \mathcal{H}$.

- The **Herbrand universe** $\mathcal{H}$ corresponding to the language $\mathcal{L}$ (or corresponding to the formula F) is defined in the following way:
  - If $c \in \mathcal{C}$ then $c \in \mathcal{H}$.
  - If $t_1, \ldots, t_n$ and $f \in \mathcal{F}$ then $f(t_1, \ldots, t_n) \in \mathcal{H}$.
  - Note that if $\mathcal{C} = \varnothing$ then add an arbitrary constant to the Herbrand universe $\mathcal{H}$.

- The **Herbrand universe** $\mathcal{H}$ corresponding to the language $\mathcal{L}$ (or corresponding to the formula F) is defined in the following way:
  - If $c \in \mathcal{C}$ then $c \in \mathcal{H}$.
  - If $t_1, \ldots, t_n$ and $f \in \mathcal{F}$ then $f(t_1, \ldots, t_n) \in \mathcal{H}$.
  - Note that if $\mathcal{C} = \varnothing$ then add an arbitrary constant to the Herbrand universe $\mathcal{H}$.
- The Herbrand universe is the set of ground terms that can be formed from the constants and function symbols of the language.

- The **Herbrand universe** $\mathcal{H}$ corresponding to the language $\mathcal{L}$ (or corresponding to the formula F) is defined in the following way:
  - If $c \in \mathcal{C}$ then $c \in \mathcal{H}$.
  - If $t_1, \ldots, t_n$ and $f \in \mathcal{F}$ then $f(t_1, \ldots, t_n) \in \mathcal{H}$.
  - Note that if $\mathcal{C} = \varnothing$ then add an arbitrary constant to the Herbrand universe $\mathcal{H}$.
- The Herbrand universe is the set of ground terms that can be formed from the constants and function symbols of the language.
- The **Herbrand base** $\mathcal{B}$ of the language $\mathcal{L}$ or the formula F is the set of ground atoms that can be formed from the predicate symbols in $\mathcal{P}$ and terms in $\mathcal{H}$.

- A **Herbrand interpretation** $\mathcal{I}_{\mathcal{H}}$ for the language $\mathcal{L}$ is an interpretation whose domain is the Herbrand universe $\mathcal{H}$ whose symbols are interpreted to themselves:

- A **Herbrand interpretation** $\mathcal{I}_{\mathcal{H}}$ for the language $\mathcal{L}$ is an interpretation whose domain is the Herbrand universe $\mathcal{H}$ whose symbols are interpreted to themselves:
  - If $c \in \mathcal{C}$, $\mathcal{I}_{\mathcal{H}}(c) = c$.

- A **Herbrand interpretation** $\mathcal{I}_{\mathcal{H}}$ for the language $\mathcal{L}$ is an interpretation whose domain is the Herbrand universe $\mathcal{H}$ whose symbols are interpreted to themselves:
  - If $c \in \mathcal{C}$, $\mathcal{I}_{\mathcal{H}}(c) = c$.
  - If $f \in \mathcal{F}$, $\mathcal{I}_{\mathcal{H}}(f) = f$.

- A **Herbrand interpretation** $\mathcal{I}_{\mathcal{H}}$ for the language $\mathcal{L}$ is an interpretation whose domain is the Herbrand universe $\mathcal{H}$ whose symbols are interpreted to themselves:
  - If $c \in \mathcal{C}$, $\mathcal{I}_{\mathcal{H}}(c) = c$.
  - If $f \in \mathcal{F}$, $\mathcal{I}_{\mathcal{H}}(f) = f$.
  - If $f \in \mathcal{P}$, $\mathcal{I}_{\mathcal{H}}(p) = p$.

- A **Herbrand interpretation** $\mathcal{I}_{\mathcal{H}}$ for the language $\mathcal{L}$ is an interpretation whose domain is the Herbrand universe $\mathcal{H}$ whose symbols are interpreted to themselves:
  - If $c \in \mathcal{C}$, $\mathcal{I}_{\mathcal{H}}(c) = c$.
  - If $f \in \mathcal{F}$, $\mathcal{I}_{\mathcal{H}}(f) = f$.
  - If $f \in \mathcal{P}$, $\mathcal{I}_{\mathcal{H}}(p) = p$.
- A **Herbrand model** for a formula (set) F is a Herbrand interpretation that satisfies F. A Herbrand model can be identified with a subset of the Herbrand base, namely the subset for which

$$\upsilon_{\mathcal{I}_{\mathcal{H}}}(p(t_1, \ldots, t_n)) = \textbf{true}.$$

Herbrand's Theorem

Several remarkable results hold.

Herbrand's Theorem

Several remarkable results hold.

Theorem

Let F be a formula. F has a model iff it has a Herbrand model.

Herbrand's Theorem

Several remarkable results hold.

Theorem

Let F be a formula. F has a model iff it has a Herbrand model.

Theorem (Herbrand's theorem (semantic form))

Let F be a formula (set). F is unsatisfiable iff a formula built from a ***finite*** *set of ground instances of subformulae of F is unsatisfiable.*

Herbrand's Theorem

Several remarkable results hold.

Theorem

Let F be a formula. F has a model iff it has a Herbrand model.

Theorem (Herbrand's theorem (semantic form))

Let F be a formula (set). F is unsatisfiable iff a formula built from a **finite** *set of ground instances of subformulae of F is unsatisfiable.*

Theorem (Herbrand's theorem (syntactic form))

A formula F is provable iff a formula built from a finite set of ground instances of subformulas of F is provable in propositional logic.

- Herbrand's theorem (semantic form) tells us that we can reduce the question of unsatisfiability in predicate logic to the question of unsatisfiability in propositional logic.

- Herbrand's theorem (semantic form) tells us that we can reduce the question of unsatisfiability in predicate logic to the question of unsatisfiability in propositional logic.
- For propositional logic, the **resolution method** is used to decide the question of satisfiability. See [Crăciun, 2010] for details.

- Herbrand's theorem (semantic form) tells us that we can reduce the question of unsatisfiability in predicate logic to the question of unsatisfiability in propositional logic.
- For propositional logic, the **resolution method** is used to decide the question of satisfiability. See [Crăciun, 2010] for details.
- For using resolution in propositional logic, propositional formulas are written in **Conjunctive Normal Form** (CNF).

- Herbrand's theorem (semantic form) tells us that we can reduce the question of unsatisfiability in predicate logic to the question of unsatisfiability in propositional logic.
- For propositional logic, the **resolution method** is used to decide the question of satisfiability. See [Crăciun, 2010] for details.
- For using resolution in propositional logic, propositional formulas are written in **Conjunctive Normal Form** (CNF).
- To use Herbrand's theorem and resolution in propositional logic, one would need a similar transformation for predicate logic.

- A **literal** in predicate logic is an atomic formula or the negation of an atomic formula.

- A **literal** in predicate logic is an atomic formula or the negation of an atomic formula.
- A formula of predicate logic is in **conjunctive normal form (CNF)** iff it is a conjunction of disjunctions of literals.

- A **literal** in predicate logic is an atomic formula or the negation of an atomic formula.
- A formula of predicate logic is in **conjunctive normal form (CNF)** iff it is a conjunction of disjunctions of literals.
- A formula of predicate logic is in **prenex conjunctive normal form (PCNF)** iff it is of the form

$$Q_1 x_1 \ldots Q_n x_n M,$$

where Q_i is a quantifier (either $\forall$, $\exists$), for $i = 1 \ldots n$, and M is a quantifier-free formula in CNF. $Q_1 x_1 \ldots Q_n x_n$ is called the **prefix** and M is called the **matrix**.

- A **literal** in predicate logic is an atomic formula or the negation of an atomic formula.
- A formula of predicate logic is in **conjunctive normal form (CNF)** iff it is a conjunction of disjunctions of literals.
- A formula of predicate logic is in **prenex conjunctive normal form (PCNF)** iff it is of the form

$$Q_1 x_1 \dots Q_n x_n M,$$

where Q_i is a quantifier (either $\forall$, $\exists$), for $i = 1 \dots n$, and M is a quantifier-free formula in CNF. $Q_1 x_1 \dots Q_n x_n$ is called the **prefix** and M is called the **matrix**.
- A formula is **closed** iff it has no free variables (i.e. all variables are bound by a quantifier).

- A **literal** in predicate logic is an atomic formula or the negation of an atomic formula.
- A formula of predicate logic is in **conjunctive normal form (CNF)** iff it is a conjunction of disjunctions of literals.
- A formula of predicate logic is in **prenex conjunctive normal form (PCNF)** iff it is of the form

$$Q_1 x_1 \dots Q_n x_n M,$$

  where Q_i is a quantifier (either $\forall$, $\exists$), for $i = 1 \dots n$, and M is a quantifier-free formula in CNF. $Q_1 x_1 \dots Q_n x_n$ is called the **prefix** and M is called the **matrix**.
- A formula is **closed** iff it has no free variables (i.e. all variables are bound by a quantifier).
- A closed formula is in **clausal form** iff it is in PCNF and its prefix consists only of universal quantifiers.

- A **literal** in predicate logic is an atomic formula or the negation of an atomic formula.
- A formula of predicate logic is in **conjunctive normal form (CNF)** iff it is a conjunction of disjunctions of literals.
- A formula of predicate logic is in **prenex conjunctive normal form (PCNF)** iff it is of the form

$$Q_1 x_1 \dots Q_n x_n M,$$

  where Q_i is a quantifier (either $\forall$, $\exists$), for $i = 1 \dots n$, and M is a quantifier-free formula in CNF. $Q_1 x_1 \dots Q_n x_n$ is called the **prefix** and M is called the **matrix**.
- A formula is **closed** iff it has no free variables (i.e. all variables are bound by a quantifier).
- A closed formula is in **clausal form** iff it is in PCNF and its prefix consists only of universal quantifiers.
- A **clause** is a disjunction of literals.

- Example: The following formula is in clausal normal form:

$$\forall x \forall y \forall z \Big(\big(p(f(x)) \vee \neg q(y,z)\big) \wedge \big(\neg p(x) \vee q(y, f(z)) \vee r(x,y)\big) \wedge \big(q(x, f(z)) \vee \neg r(f(y), f(z))\big) \Big).$$

- Example: The following formula is in clausal normal form:

$$\forall x \forall y \forall z \Big((p(f(x)) \vee \neg q(y,z)) \wedge$$
$$(\neg p(x) \vee q(y, f(z)) \vee r(x,y)) \wedge$$
$$(q(x, f(z)) \vee \neg r(f(y), f(z))) \Big).$$

- Notation: Since the matrix only consists of universal quantifiers, these can be omitted. The clausal form can be represented in the following manner (clauses as **sets of literals**, formulae in clausal form as **sets of clauses**):

$$\Big\{ \{p(f(x)), \neg q(y,z)\},$$
$$\{\neg p(x), q(y, f(z)), r(x,y)\},$$
$$\{q(x, f(z)), \neg r(f(y), f(z))\} \Big\}.$$

- Example: The following formula is in clausal normal form:

$$\forall x \forall y \forall z \Big((p(f(x)) \vee \neg q(y,z)) \wedge \\ (\neg p(x) \vee q(y, f(z)) \vee r(x,y)) \wedge \\ (q(x, f(z)) \vee \neg r(f(y), f(z))) \Big).$$

- Notation: Since the matrix only consists of universal quantifiers, these can be omitted. The clausal form can be represented in the following manner (clauses as **sets of literals**, formulae in clausal form as **sets of clauses**):

$$\Big\{ \{p(f(x)), \neg q(y,z)\}, \\ \{\neg p(x), q(y, f(z)), r(x,y)\}, \\ \{q(x, f(z)), \neg r(f(y), f(z))\} \Big\}.$$

- Notation: Let F, G be formulas. We denote $F \approx G$ if F and G are equisatisfiable (i.e. F is satisfiable iff G is satisfiable).

Theorem (Skolem)

Let F be a closed formula. Then there exists a formula F' in clausal form such that $F \approx F'$.

- Skolem's theorem can be used to decide if a formula is unsatisfiable if a method for deciding unsatisfiability of formulas in clausal exists.

Skolemization Algorithm

IN: closed formula F.

Skolemization Algorithm

IN: closed formula F.

OUT: formula F' in clausal form such that $F \approx F'$.

Skolemization Algorithm

IN: closed formula F.

OUT: formula F' in clausal form such that $F \approx F'$.

Running example:

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x))$$

Skolemization Algorithm

IN: closed formula F.

OUT: formula F' in clausal form such that $F \approx F'$.

Running example:

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x))$$

1: Rename the bound variables such that no variable appears in the scope of two different quantifiers.

Skolemization Algorithm

IN: closed formula F.

OUT: formula F' in clausal form such that $F \approx F'$.

Running example:

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x))$$

1: Rename the bound variables such that no variable appears in the scope of two different quantifiers.

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall y p(y) \Rightarrow \forall z q(z))$$

Skolemization Algorithm

IN: closed formula F.

OUT: formula F' in clausal form such that $F \approx F'$.

Running example:

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x))$$

1: Rename the bound variables such that no variable appears in the scope of two different quantifiers.

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall y p(y) \Rightarrow \forall z q(z))$$

2: Eliminate all the equivalence and implication connectives $(\Leftrightarrow, \Rightarrow)$.

Skolemization Algorithm

IN: closed formula F.

OUT: formula F' in clausal form such that $F \approx F'$.

Running example:

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x))$$

1: Rename the bound variables such that no variable appears in the scope of two different quantifiers.

$$\forall x(p(x) \Rightarrow q(x)) \Rightarrow (\forall y p(y) \Rightarrow \forall z q(z))$$

2: Eliminate all the equivalence and implication connectives $(\Leftrightarrow, \Rightarrow)$.

$$\neg\forall x(\neg p(x) \vee q(x)) \vee (\neg\forall y p(y) \vee \forall z q(z))$$

3: Push the negations inside the parantheses, until negations apply only to atomic formulae. Use the equivalences

$$
\begin{aligned}
&\neg(\neg F) \equiv F, \\
&\neg(F \wedge G) \equiv (\neg F \vee \neg G), \\
&\neg(F \vee G) \equiv (\neg F \wedge \neg G), \\
&\neg(\forall x F[x]) \equiv \exists x \neg F[x], \\
&\neg(\exists x F[x]) \equiv \forall x \neg F[x].
\end{aligned}
$$

3: Push the negations inside the parantheses, until negations apply only to atomic formulae. Use the equivalences

$$\begin{aligned}
\neg(\neg F) &\equiv F, \\
\neg(F \wedge G) &\equiv (\neg F \vee \neg G), \\
\neg(F \vee G) &\equiv (\neg F \wedge \neg G), \\
\neg(\forall x F[x]) &\equiv \exists x \neg F[x], \\
\neg(\exists x F[x]) &\equiv \forall x \neg F[x].
\end{aligned}$$

$$\exists x (p(x) \wedge \neg q(x)) \vee \exists y \neg p(y) \vee \forall z q(z)$$

3: Push the negations inside the parantheses, until negations apply only to atomic formulae. Use the equivalences

$$
\begin{aligned}
&\neg(\neg F) \equiv F,\\
&\neg(F \wedge G) \equiv (\neg F \vee \neg G),\\
&\neg(F \vee G) \equiv (\neg F \wedge \neg G),\\
&\neg(\forall x F[x]) \equiv \exists x \neg F[x],\\
&\neg(\exists x F[x]) \equiv \forall x \neg F[x].
\end{aligned}
$$

$$\exists x(p(x) \wedge \neg q(x)) \vee \exists y \neg p(y) \vee \forall z q(z)$$

4: Extract the quantifiers from the matrix. Since the variables have been renamed, the following equivalences can be applied: $AopQxB[x] \equiv Qx(AopB[X])$ and $QxB[x]opA \equiv Qx(B[X]opA)$ where Q is one of $\forall, \exists$ and op is one of $\wedge, \vee$.

3: Push the negations inside the parantheses, until negations apply only to atomic formulae. Use the equivalences

$$\begin{aligned}
\neg(\neg F) &\equiv F, \\
\neg(F \wedge G) &\equiv (\neg F \vee \neg G), \\
\neg(F \vee G) &\equiv (\neg F \wedge \neg G), \\
\neg(\forall x F[x]) &\equiv \exists x \neg F[x], \\
\neg(\exists x F[x]) &\equiv \forall x \neg F[x].
\end{aligned}$$

$$\exists x(p(x) \wedge \neg q(x)) \vee \exists y \neg p(y) \vee \forall z q(z)$$

4: Extract the quantifiers from the matrix. Since the variables have been renamed, the following equivalences can be applied: $AopQxB[x] \equiv Qx(AopB[X])$ and $QxB[x]opA \equiv Qx(B[X]opA)$ where Q is one of $\forall, \exists$ and op is one of $\wedge, \vee$.

$$\exists x \exists y \forall z((p(x) \wedge \neg q(x)) \vee \neg p(y) \vee q(z))$$

5: Use the distributive laws $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, $(P \wedge Q) \vee R) \equiv (P \vee R) \wedge (Q \vee R)$ to transform the matrix into CNF.

Note that steps 1-5 preserve logical consequence. It is relatively easy to show that step 6 preserves satisfiability. For details,

5: Use the distributive laws $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, $(P \wedge Q) \vee R) \equiv (P \vee R) \wedge (Q \vee R)$ to transform the matrix into CNF.

$$\exists x \exists y \forall z ((p(x) \vee \neg p(y) \vee q(z)) \wedge (\neg q(x) \vee \neg p(y) \vee q(z)))$$

Note that steps 1-5 preserve logical consequence. It is relatively easy to show that step 6 preserves satisfiability. For details,

5: Use the distributive laws $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, $(P \wedge Q) \vee R) \equiv (P \vee R) \wedge (Q \vee R)$ to transform the matrix into CNF.

$$\exists x \exists y \forall z((p(x) \vee \neg p(y) \vee q(z)) \wedge (\neg q(x) \vee \neg p(y) \vee q(z)))$$

6: **Skolemization**

Note that steps 1-5 preserve logical consequence. It is relatively easy to show that step 6 preserves satisfiability. For details,

5: Use the distributive laws $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, $(P \wedge Q) \vee R) \equiv (P \vee R) \wedge (Q \vee R)$ to transform the matrix into CNF.

$$\exists x \exists y \forall z ((p(x) \vee \neg p(y) \vee q(z)) \wedge (\neg q(x) \vee \neg p(y) \vee q(z)))$$

6: **Skolemization**

- If the prefix is of the form $\forall y_1 \ldots \forall y_n \exists x$, let f be a new n-ary function symbol. Delete $\exists x$ from the prefix, replace all occurences of x in the matrix by $f(y_1, \ldots, y_n)$. The function f is called a **Skolem function**.

Note that steps 1-5 preserve logical consequence. It is relatively easy to show that step 6 preserves satisfiability. For details,

5: Use the distributive laws $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, $(P \wedge Q) \vee R) \equiv (P \vee R) \wedge (Q \vee R)$ to transform the matrix into CNF.

$$\exists x \exists y \forall z((p(x) \vee \neg p(y) \vee q(z)) \wedge (\neg q(x) \vee \neg p(y) \vee q(z)))$$

6: **Skolemization**

- If the prefix is of the form $\forall y_1 \ldots \forall y_n \exists x$, let f be a new n-ary function symbol. Delete $\exists x$ from the prefix, replace all occurences of x in the matrix by $f(y_1, \ldots, y_n)$. The function f is called a **Skolem function**.
- If there are no universal quantifiers before $\exists x$ in the prefix, let a be a new constant. Eliminate $\exists x$ from the prefix and replace every occurence of x in the matrix with a. The constant a is a **Skolem constant**.

Note that steps 1-5 preserve logical consequence. It is relatively easy to show that step 6 preserves satisfiability. For details,

5: Use the distributive laws $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$, $(P \wedge Q) \vee R) \equiv (P \vee R) \wedge (Q \vee R)$ to transform the matrix into CNF.

$$\exists x \exists y \forall z ((p(x) \vee \neg p(y) \vee q(z)) \wedge (\neg q(x) \vee \neg p(y) \vee q(z)))$$

6: **Skolemization**

- If the prefix is of the form $\forall y_1 \ldots \forall y_n \exists x$, let f be a new n-ary function symbol. Delete $\exists x$ from the prefix, replace all occurences of x in the matrix by $f(y_1, \ldots, y_n)$. The function f is called a **Skolem function**.
- If there are no universal quantifiers before $\exists x$ in the prefix, let a be a new constant. Eliminate $\exists x$ from the prefix and replace every occurence of x in the matrix with a. The constant a is a **Skolem constant**.

$$\forall z ((p(a) \vee \neg p(b) \vee q(z)) \wedge (\neg q(a) \vee \neg p(b) \vee q(z)))$$

Note that steps 1-5 preserve logical consequence. It is relatively easy to show that step 6 preserves satisfiability. For details,

- Herbrand's theorem reduces the problem of establishing unsatisfiability of a formula (set) to the problem of establishing unsatisfiability of a finite set of ground formulae.

- Herbrand's theorem reduces the problem of establishing unsatisfiability of a formula (set) to the problem of establishing unsatisfiability of a finite set of ground formulae.
- For practical purposes, given a finite set of ground formulae, one can rename the distinct ground atoms by distinct propositional formulae and thus answer the question of unsatisfiability by **propositional resolution**.
  See [Crăciun, 2010] for details on propositional resolution.

- Herbrand's theorem reduces the problem of establishing unsatisfiability of a formula (set) to the problem of establishing unsatisfiability of a finite set of ground formulae.
- For practical purposes, given a finite set of ground formulae, one can rename the distinct ground atoms by distinct propositional formulae and thus answer the question of unsatisfiability by **propositional resolution**.
  See [Crăciun, 2010] for details on propositional resolution.
- However, this approach is not practical: there is no indication how to find the finite set of ground formulae: the set of possible ground instantiations is both unbounded and unstructured.

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.

- An **expression** is a term or formula (in particular literal, clause, or set of clauses).

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \dots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.
- An **expression** is a term or formula (in particular literal, clause, or set of clauses).
- Let E be an expression, $\theta = \{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}$. An **instance of** E (or the result of applying θ to E), $E\theta$ is the expression obtained by **simultaneously** replacing every occurrence of x_i in E by t_i.

- A **substitution** of terms for variables is a set:

$$\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

  where, for $i = 1 \ldots n$, x_i are distinct variables, t_i are terms such that x_i and t_i are distinct. Substitutions will be denoted by lowercase greek letters (λ, θ, δ,σ). The **empty** substitution is denoted by ϵ.
- An **expression** is a term or formula (in particular literal, clause, or set of clauses).
- Let E be an expression, $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. An **instance of** E (or the result of applying θ to E), $E\theta$ is the expression obtained by **simultaneously** replacing every occurrence of x_i in E by t_i.
- Example: let $E = p(x) \vee q(f(y))$, $\theta = \{x \leftarrow y, y \leftarrow f(a)\}$. Then

$$E\theta = p(y) \vee q(f(f(a))).$$

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ,** $\theta\sigma$ is the substitution:

  $$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ, $\theta\sigma$** is the substitution:

$$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.
- Example: let

$$\theta = \{x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u\},$$
$$\sigma = \{y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))\},$$

  then:

$$\theta\sigma = \{x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))\}.$$

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ, $\theta\sigma$** is the substitution:

  $$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.
- Example: let

  $$\theta = \{x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u\},$$
  $$\sigma = \{y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))\},$$

  then:

  $$\theta\sigma = \{x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))\}.$$
- Let E be an expression and θ, σ substitutions. Then $E(\theta\sigma) = (E\theta)\sigma$.

- Let $\theta = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ and $\sigma = \{y_1 \leftarrow s_1, \ldots, y_k \leftarrow s_k\}$ be substitutions. Let X, Y be the sets of variables from θ and σ, respectively. The **composition of θ and σ, $\theta\sigma$** is the substitution:

  $$\theta\sigma = \{x_i \leftarrow t_i\sigma | x_i \in X, x_i \neq t_i\sigma\} \cup \{y_j \leftarrow s_j | y_j \in Y, y_j \notin X\},$$

  in other words, apply the substitution σ to the terms t_i (provided that the resulting substitution does not collapse into $x_i \leftarrow x_i$) then append the substitutions from σ whose variables do not appear already in θ.
- Example: let

  $$\begin{aligned} \theta &= \{x \leftarrow f(y), y \leftarrow f(a), z \leftarrow u\}, \\ \sigma &= \{y \leftarrow g(a), u \leftarrow z, v \leftarrow f(f(a))\}, \end{aligned}$$

  then:

  $$\theta\sigma = \{x \leftarrow f(g(a)), y \leftarrow f(a), u \leftarrow z, v \leftarrow f(f(a))\}.$$

- Let E be an expression and θ, σ substitutions. Then $E(\theta\sigma) = (E\theta)\sigma$.
- Let θ, σ, λ be substitutions. Then $\theta(\sigma\lambda) = (\theta\sigma)\lambda$.

Unifiers

- Consider two nonground literals: $p(f(x), g(y))$ and $p(f(f(a)), g(z))$:

Unifiers

- Consider two nonground literals: $p(f(x), g(y))$ and $p(f(f(a)), g(z))$:
  - the substitution

    $$\{x \leftarrow f(a), y \leftarrow f(g(a)), z \leftarrow f(g(a))\}$$

    applied to both the literals will make them identical (will "unify" them),

Unifiers

- Consider two nonground literals: $p(f(x), g(y))$ and $p(f(f(a)), g(z))$:
  - the substitution

    $$\{x \leftarrow f(a), y \leftarrow f(g(a)), z \leftarrow f(g(a))\}$$

    applied to both the literals will make them identical (will "unify" them),
  - the same effect is obtained when applying the substitutions

    $$\{x \leftarrow f(a), y \leftarrow a, z \leftarrow a\},$$

    $$\{x \leftarrow f(a), z \leftarrow y\}.$$

- Given a set of literals, a **unifier** is a substitution that makes the atoms of the set identical. A **most general unifier (mgu)** is a unifier μ such that any other unifier θ can be obtained from μ by a further substitution λ such that $\theta = \mu\lambda$.

- Given a set of literals, a **unifier** is a substitution that makes the atoms of the set identical. A **most general unifier (mgu)** is a unifier μ such that any other unifier θ can be obtained from μ by a further substitution λ such that $\theta = \mu\lambda$.
- Note that not all literals are unifiable: if the predicate symbols are different, the literals cannot be unified. Also, consider the case of $p(x)$ and $p(f(x))$. Since the substitution of the variable x has to be done in the same time, the terms x and $f(x)$ cannot be made identical, and the unification will fail.

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$\begin{aligned} f(x) &= f(f(a)) \\ g(y) &= g(z). \end{aligned}$$

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$\begin{aligned} f(x) &= f(f(a)) \\ g(y) &= g(z). \end{aligned}$$

- A set of term equations is in **solved form** iff:

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$
\begin{aligned}
f(x) &= f(f(a)) \\
g(y) &= g(z).
\end{aligned}
$$

- A set of term equations is in **solved form** iff:
  - all equations are of the form $x_i = t_i$, where x_i are variables,

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$f(x) = f(f(a))$$
$$g(y) = g(z).$$

- A set of term equations is in **solved form** iff:
  - all equations are of the form $x_i = t_i$, where x_i are variables,
  - each variable x_i that appears on the left hand side of an equation does not appear elsewhere in a set.

Unification algorithm

- Note that the unifiability of the literals $p(f(x), g(y))$ and $p(f(f(a)), g(z))$ can be expressed as a set of term equations:

$$\begin{aligned} f(x) &= f(f(a)) \\ g(y) &= g(z). \end{aligned}$$

- A set of term equations is in **solved form** iff:
  - all equations are of the form $x_i = t_i$, where x_i are variables,
  - each variable x_i that appears on the left hand side of an equation does not appear elsewhere in a set.
    A set of equations in solved form defines a substitution in a natural way by turning each equation $x_i = t_i$ into an element of the substitution, $x_i \leftarrow t_i$.

Unification Algorithm

INPUT: A set of term equations.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.
2. Erase the equation $x = x$, for all x, variables.

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.
2. Erase the equation $x = x$, for all x, variables.
3. Let $t' = t''$ be an equation where t', t'' are not variables. If the outermost (function) symbol of t' and t'' are not identical, terminate and answer "not unifiable". Otherwise, if t' is of the form $f(t'_1, \ldots, t'_k)$ and t'' is of the form $f(t''_1, \ldots, t''_k)$, replace the equation $f(t'_1, \ldots, t'_k) = f(t''_1, \ldots, t''_k)$ by the k equations
$$t'_1 = t''_1, \ldots, t'_k = t''_k.$$

Unification Algorithm

INPUT: A set of term equations.

OUTPUT: A set of term equations in solved form, or "not unifiable".

Perform the following transformations on the set of equations as long as any of them can still be performed:

1. Transform $t = x$ into $x = t$, where x is a variable and t is not.
2. Erase the equation $x = x$, for all x, variables.
3. Let $t' = t''$ be an equation where t', t'' are not variables. If the outermost (function) symbol of t' and t'' are not identical, terminate and answer "not unifiable". Otherwise, if t' is of the form $f(t'_1, \dots, t'_k)$ and t'' is of the form $f(t''_1, \dots, t''_k)$, replace the equation $f(t'_1, \dots, t'_k) = f(t''_1, \dots, t''_k)$ by the k equations
$$t'_1 = t''_1, \dots, t'_k = t''_k.$$
4. Let $x = t$ a term equation such that x has another occurrence in the set of term equations. If x occurs in t (occurs check!), terminate and answer "not unifiable". Otherwise, transform the equation set by replacing each occurrence of x in other equations by t.

Example from [Ben-Ari, 2001]

Consider the following two equations:

$$g(y) = x$$
$$f(x, h(x), y) = f(g(z), w, z).$$

- Apply rule 1 to the first equation and rule 3 to the second equation:

$$x = g(y)$$
$$x = g(z)$$
$$h(x) = w$$
$$y = z.$$

- Apply rule 4 on the second equation to replace the other occurrences of x:
$$\begin{aligned} g(z) &= g(y) \\ x &= g(z) \\ h(g(z)) &= w \\ y &= z. \end{aligned}$$
- Apply rule 3 to the first equation
$$\begin{aligned} z &= y \\ x &= g(z) \\ h(g(z)) &= w \\ y &= z. \end{aligned}$$

- Apply rule 4 on the last equation to replace y by z in the first equation, then erase the resulting $z = z$ using rule 2:

$$\begin{aligned} x &= g(z) \\ h(g(z)) &= w \\ y &= z. \end{aligned}$$

- Transform the second equation by rule 1:

$$\begin{aligned} x &= g(z) \\ w &= h(g(z)) \\ y &= z. \end{aligned}$$

- The algorithm terminates successfully. The resulting substitution

$$\{x \leftarrow g(z), w \leftarrow h(g(z)), y \leftarrow z\}$$

is the most general unifier of the initial set of equations.

Theorem (Correctness of the unification algorithm)

The unification algorithm terminates. If the algorithm terminates with the answer "not unifiable", there is no unifier for the set of term equations. If it terminates successfully, the resulting set of equations is in solved form and it defines an mgu

$$\mu = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

of the set of equations

Proof.

See [Ben-Ari, 2001], pp. 158. □

- Read: Chapter 7, sections 7.5-7.8 of [Ben-Ari, 2001]; Chapter 8, sections 8.1-8.3 of [Ben-Ari, 2001].
- Also read: Chapter 2, Chapter 3 of [Nilsson and Maluszynski, 2000].

Ben-Ari, M. (2001).
Mathematical Logic for Computer Science.
Springer Verlag, London, 2nd edition.

Crăciun, A. (2005-2010).
Logic for Computer Science.

Nilsson, U. and Maluszynski, J. (2000).
Logic, Programming and Prolog.
copyright Ulf Nilsson and Jan Maluszynski, 2nd edition.