FIRST-ORDER LOGIC:
SYNTAX AND SEMANTICS

Course “Computational Logic”

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at
Abstract Syntax

A first-order formula $F$ is a “sentence” that talks about “objects”.

Sentence: “The successor of every natural number has a natural number as its predecessor.”

Formula: $\forall x. \text{isNat}(x) \Rightarrow \exists y. \text{isNat}(y) \land \text{isPred}(y, \text{succ}(x))$

Two kinds of syntactic phrases (“expressions”):

- **Terms** denoting objects (values).
  
  $$t ::= x \mid c \mid f(t_1, \ldots, t_n)$$

- **Formulas** denoting properties of objects (i.e., the truth values “true” or “false”).
  
  $$F ::= \top \mid \bot \mid p(t_1, \ldots, t_n) \mid \ldots \mid (\forall x. F) \mid (\exists x. F)$$

  - Variables $x \in X$.
  - Constants $c \in C$, $n$-ary function symbols $f \in F$ and predicate symbols $p \in P$.
  - Quantifiers $\forall$ (“universal quantifier”) and $\exists$ (“existential quantifier”).
    - $\forall x. F$: “for all (possible objects assigned to) $x$, $F$ is true”.
    - $\exists x. F$: “there exists some (possible object assigned to) $x$, for which $F$ is true”.

Other versions of concrete syntax can be transformed into above “standard form”.

1/34
Concrete Syntax

- We use the following abbreviations:

\[
\forall x_1, \ldots, x_n. F \rightarrow \forall x_1. \ldots \forall x_n. F
\]
\[
\exists x_1, \ldots, x_n. F \rightarrow \exists x_1. \ldots \exists x_n. F
\]

- We apply the following binding rules:

\[
(\neg) > \ldots > (\forall, \exists)
\]

- Without parentheses, the scope of a quantifiers ranges till the *end* of the formula.

\[
\forall x. \text{isNat}(x) \Rightarrow \exists y. \text{isNat}(y) \land \text{isPred}(y, \text{succ}(x))
\]
\[
\sim \forall x. (\text{isNat}(x) \Rightarrow \exists y. (\text{isNat}(y) \land \text{isPred}(y, \text{succ}(x))))
\]

Be sure to (mentally) insert parentheses appropriately.
Abstract Syntax in OCaml

- **OCaml Type:**

  ```ocaml
type term = Var of string | Fn of string * term list;;
  type ('a)formula = False | True | Atom of 'a
  | Not of ('a)formula | And of ('a)formula * ('a)formula | Or of ('a)formula * ('a)formula
  | Imp of ('a)formula * ('a)formula | Iff of ('a)formula * ('a)formula
  | Forall of string * ('a)formula | Exists of string * ('a)formula;;
  type fol = R of string * term list;;
  type folformula = fol formula;;
  ```

- **Execution:**

  ```ocaml
  # let f = << forall x. P(x) ==> exists y. R(x,F(x,y)) >> ;;
  val f : fol formula = <<forall x. P(x) ==> (exists y. R(x,F(x,y)))>>

  let g = Forall("x", Imp(Atom(R("P", [Var "x"])),
       Exists("y", Atom(R("R", [Var "x"; Fn("F", [Var "x"; Var "y"])])))))) ;;
  val g : fol formula = <<forall x. P(x) ==> (exists y. R(x,F(x,y)))>>
  ```

First-order formulas are values of type **fol formula**.
Interpretation of Quantifiers

- Formula $E$: “everybody loves somebody”
  \[ \forall x. \exists y. \text{loves}(x, y) \]

- Formula $S$: “somebody is loved by everybody”
  \[ \exists y. \forall x. \text{loves}(x, y) \]

The nesting order of quantifier matters.

$E$: true, $S$: false

$E$: true, $S$: true
Free and Bound Variables

- **Non-closed formula:**
  \[ \text{equal}(x, \text{zero}) \]
  - Truth value depends on value assigned to \( x \): “true” for \( x = \text{zero} \), “false”, otherwise.
  - Variable \( x \) is free in the formula.
  - If some of its variables are free, a formula is non-closed.

- **Closed formulas (sentences):**
  \[ \forall x. \text{equal}(x, \text{zero}) \quad \exists x. \text{equal}(x, \text{zero}) \]
  - Truth values do not depend on \( x \): first formula is “false”, second one is “true”.
  - Variable \( x \) is bound in both formulas (by the quantifier \( \forall \) respectively \( \exists \)).
  - If all of its variables are bound, a formula is closed.

The truth value of a formula only depends on the values assigned to the formula’s free variables; it is independent of the values of the bound variables.
The Set of Free Variables

fv($F$) and fv($t$) compute the set of free vars of formula $F$ and term $t$.

$$
\begin{align*}
\text{fv}(\tau) &= \emptyset \\
\text{fv}(\bot) &= \emptyset \\
\text{fv}(p(t_1, \ldots, t_n)) &= \text{fv}(t_1) \cup \ldots \cup \text{fv}(t_n) \\
\text{fv}(\neg F) &= \text{fv}(F) \\
\text{fv}(F_1 \land F_2) &= \text{fv}(F_1) \cup \text{fv}(F_2) \\
\text{fv}(F_1 \lor F_2) &= \text{fv}(F_1) \cup \text{fv}(F_2) \\
\text{fv}(F_1 \Rightarrow F_2) &= \text{fv}(F_1) \cup \text{fv}(F_2) \\
\text{fv}(F_1 \Leftrightarrow F_2) &= \text{fv}(F_1) \cup \text{fv}(F_2) \\
\text{fv}(\forall x. \ F) &= \text{fv}(F) \setminus \{x\} \\
\text{fv}(\exists x. \ F) &= \text{fv}(F) \setminus \{x\}
\end{align*}
$$

Quantifiers bind variables.

fv($x$) = $\{x\}$  fv($c$) = $\emptyset$

fv($f(t_1, \ldots, t_n)$) = $\text{fv}(t_1) \cup \ldots \cup \text{fv}(t_n)$

Example

$$
\begin{align*}
\text{fv}(q(x, y, z)) &= \{x, y, z\} \\
\text{fv}(\exists y. \ q(x, y, z)) &= \text{fv}(q(x, y, z)) \setminus \{y\} \\
&= \{x, y, z\} \setminus \{y\} = \{x, z\} \\
\text{fv}(p(x, w)) &= \{x, w\} \\
\text{fv}(p(x, w) \Rightarrow \exists y. \ q(x, y, z)) &= \text{fv}(p(x, w)) \cup \text{fv}(\exists y. \ q(x, y, z)) \\
&= \{x, w\} \cup \{x, z\} = \{x, w, z\} \\
\text{fv}(\forall x. \ p(x, w) \Rightarrow \exists y. \ q(x, y, z)) &= \text{fv}(p(x, w) \Rightarrow \exists y. \ q(x, y, z)) \setminus \{x\} \\
&= \{x, w, z\} \setminus \{x\} = \{w, z\}
\end{align*}
$$
let rec fvt tm = 
  match tm with 
   Var x -> [x] 
 | Fn(f,args) -> unions (map fvt args);;

let rec fv fm = 
  match fm with 
   False | True -> [] 
 | Atom(R(p,args)) -> unions (map fvt args) 
 | Not(p) -> fv p 
 | And(p,q) | Or(p,q) | Imp(p,q) | Iff(p,q) -> union (fv p) (fv q) 
 | Forall(x,p) | Exists(x,p) -> subtract (fv p) [x];;

let generalize fm = itlist mk_forall (fv fm) fm;;

# fv <<forall x. p(x,w) ==> exists y. q(x,y,z)>>;;
- : string list = ["w"; "z"]
# generalize <<forall x. p(x,w) ==> exists y. q(x,y,z)>>;;
- : fol formula = <<forall w z x. p(x,w) ==> (exists y. q(x,y,z))>>
Formal Semantics: Structures and Valuations

- A structure \((D, I)\) consists of a domain \(D\) and an interpretation \(I\) on \(D\):
  - \(D\) is a non-empty set of objects \((D \neq \emptyset)\).
    - The “universe” about which a first-order formula talks.
  - \(I\) maps every constant and function/predicate symbol to its meaning:
    - Constant \(c \in C\): \(I(c)\) is an object in \(D\) \((I(c) \in D)\).
    - Function symbol \(f \in F\) of arity \(n\): \(I(f)\) is an \(n\)-ary function on \(D\) \((I(f): D^n \to D)\).
    - Predicate symbol \(p \in P\) of arity \(n\): \(I(p)\) is an \(n\)-ary predicate/relation on \(D\) \((I(p) \subseteq D^n)\).

- A valuation (assignment) \(v\) maps every variable to its meaning:
  - Variable \(x \in X\): \(v(x)\) is an object in \(D\) \((v(x) \in D)\).

\[
D = \mathbb{N} \\
I = [0 \leftrightarrow \text{zero}, + \leftrightarrow \text{add}, < \leftrightarrow \text{less-than}, \ldots] \\
v = [x \leftrightarrow \text{one}, y \leftrightarrow \text{zero}, z \leftrightarrow \text{three}, \ldots]
\]
The Formal Semantics: Terms

- Term semantics $\llbracket t \rrbracket^{D,I}_v \in D$
  - Given structure $(D, I)$ and valuation $v$, the semantics of term $t$ is an object in $D$.
    $$t ::= x \mid c \mid f(t_1, \ldots, t_n)$$
  - The meaning of a variable is the value given by the valuation:
    $$\llbracket x \rrbracket^{D,I}_v := v(x)$$
  - The meaning of a constant is the value given by the interpretation:
    $$\llbracket c \rrbracket^{D,I}_v := I(c)$$
  - The meaning of a function application is the result of the interpretation of the function symbol applied to the values of the argument terms:
    $$\llbracket f(t_1, \ldots, t_n) \rrbracket^{D,I}_v := I(f)(\llbracket t_1 \rrbracket^{D,I}_v, \ldots, \llbracket t_n \rrbracket^{D,I}_v)$$

The recursive definition of a function evaluating a term.
Example

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

\[
\lfloor x + (y + 0) \rfloor^D_v I = \text{add}(\lfloor x \rfloor_v^D I, \lfloor y + 0 \rfloor_v^D I)
\]
\[
= \text{add}(\nu(x), \lfloor y + 0 \rfloor_v^D I)
\]
\[
= \text{add}(\text{one}, \lfloor y + 0 \rfloor_v^D I)
\]
\[
= \text{add}(\text{one}, \text{add}(\lfloor y \rfloor_v^D I, \lfloor 0 \rfloor_v^D I))
\]
\[
= \text{add}(\text{one}, \text{add}(\nu(y), I(0)))
\]
\[
= \text{add}(\text{one}, \text{add}(\text{two, zero}))
\]
\[
= \text{add}(\text{one, two})
\]
\[
= \text{three.}
\]

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

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

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

The meaning of the term with another interpretation.
Formal Semantics: Basic Formulas

- Formula semantics $\mathfrak{I}_v^D, I \in \mathbb{B}$
  - Given structure $(D, I)$ and valuation $v$, the semantics of formula $F$ is a truth value.
    \[ F ::= \top | \bot | p(t_1, \ldots, t_n) | \ldots | (\forall x. F) | (\exists x. F) \]
  - The meaning of the logical constants is a fixed truth value:
    \[ \mathfrak{I}_v^D, I : \top ::= \text{true} \quad \mathfrak{I}_v^D, I : \bot ::= \text{false} \]
  - The meaning of an atomic formula is the result of the interpretation of the predicate symbol applied to the values of the argument terms.
    \[ \mathfrak{I}_v^D, I : p(t_1, \ldots, t_n) ::= I(p)(\mathfrak{I}_v^D, t_1, \ldots, \mathfrak{I}_v^D, t_n) \]

The meaning of the basic formulas.
Formal Semantics: Logical Connectives

- The meaning of the logical connectives:

\[
\begin{align*}
[\neg F]_{D,I}^v &= \begin{cases} 
\text{true} & \text{if } [F]_{D,I}^v = \text{false} \\
\text{false} & \text{else}
\end{cases} \\
[ F_1 \wedge F_2 ]_{D,I}^v &= \begin{cases} 
\text{true} & \text{if } [F_1]_{D,I}^v = [F_2]_{D,I}^v = \text{true} \\
\text{false} & \text{else}
\end{cases} \\
[ F_1 \vee F_2 ]_{D,I}^v &= \begin{cases} 
\text{false} & \text{if } [F_1]_{D,I}^v = [F_2]_{D,I}^v = \text{false} \\
\text{true} & \text{else}
\end{cases} \\
[ F_1 \Rightarrow F_2 ]_{D,I}^v &= \begin{cases} 
\text{false} & \text{if } [F_1]_{D,I}^v = \text{true} \text{ and } [F_2]_{D,I}^v = \text{false} \\
\text{true} & \text{else}
\end{cases} \\
[ F_1 \Leftrightarrow F_2 ]_{D,I}^v &= \begin{cases} 
\text{true} & \text{if } [F_1]_{D,I}^v = [F_2]_{D,I}^v \\
\text{false} & \text{else}
\end{cases}
\end{align*}
\]

An embedding of the semantics of propositional logic into first-order logic.
Formal Semantics: Quantifiers

- $(\forall x. F)$ is true, if $F$ is true for every possible object $d$ assigned to variable $x$:

$$\llbracket \forall x. F \rrbracket^D_I := \begin{cases} \text{true} & \text{if } \llbracket F \rrbracket^D_I = \text{true for all } d \text{ in } D \\ \text{false} & \text{else} \end{cases}$$

- $(\exists x. F)$ is true, if $F$ is true for at least one possible object $d$ assigned to $x$:

$$\llbracket \exists x. F \rrbracket^D_I := \begin{cases} \text{true} & \text{if } \llbracket F \rrbracket^D_I = \text{true for some } d \text{ in } D \\ \text{false} & \text{else} \end{cases}$$

- Valuation $v$ updated by the assignment of object $d$ to variable $x$:

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

The core of the semantics of first-order logic.
Example

\[ D = \mathbb{N}_3 = \{ \text{zero, one, two} \} \quad I = [0 \leftrightarrow \text{zero}, + \leftrightarrow \text{add}, \ldots] \quad v = [x \mapsto \text{one}, y \mapsto \text{two}, z \mapsto \text{two}, \ldots] \]

\[
\left[ \forall x. \exists y. x + y = z \right]^{D, I}_{v} = ?
\]

- \[
\left[ \exists y. x + y = z \right]^{D, I}_{v[x \mapsto \text{zero}]} = \text{true}
\]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{zero}, y \mapsto \text{zero}]} = \text{false}
  \]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{zero}, y \mapsto \text{one}]} = \text{false}
  \]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{zero}, y \mapsto \text{two}]} = \text{true}
  \]

- \[
\left[ \exists y. x + y = z \right]^{D, I}_{v[x \mapsto \text{one}]} = \text{true}
\]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{one}, y \mapsto \text{zero}]} = \text{false}
  \]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{one}, y \mapsto \text{one}]} = \text{true}
  \]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{one}, y \mapsto \text{two}]} = \text{false}
  \]

- \[
\left[ \exists y. x + y = z \right]^{D, I}_{v[x \mapsto \text{two}]} = \text{true}
\]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{two}, y \mapsto \text{zero}]} = \text{true}
  \]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{two}, y \mapsto \text{one}]} = \text{false}
  \]
  \[
  \left[ x + y = z \right]^{D, I}_{v[x \mapsto \text{two}, y \mapsto \text{two}]} = \text{false}
  \]

\[
\left[ \forall x. \exists y. x + y = z \right]^{D, I}_{v} = \text{true}.
\]
Term and Formula Semantics in OCaml

```
let rec termval (domain,func,pred as m) v tm =
  match tm with
  | Var(x) -> apply v x
  | Fn(f,args) -> func f (map (termval m v) args);;

let rec holds (domain,func,pred as m) v fm =
  match fm with
  | False -> false
  | True -> true
  | Atom(R(r,args)) -> pred r (map (termval m v) args)
  | Not(p) -> not(holds m v p)
  | And(p,q) -> (holds m v p) & (holds m v q)
  | Or(p,q) -> (holds m v p) or (holds m v q)
  | Imp(p,q) -> not(holds m v p) or (holds m v q)
  | Iff(p,q) -> (holds m v p = holds m v q)
  | Forall(x,p) -> forall (fun a -> holds m ((x |-> a) v) p) domain
  | Exists(x,p) -> exists (fun a -> holds m ((x |-> a) v) p) domain;;
```

The structure is represented by a triple \( m = (\text{domain}, \text{func}, \text{pred}) \).
let bool_interp =
  let func f args =
      match (f, args) with
      | ("0",[]) -> false
      | ("1",[]) -> true
      | ("+",[x;y]) -> not(x = y)
      | ("*",[x;y]) -> x & y
      | _ -> failwith "uninterpreted function"
  and pred p args =
      match (p, args) with
      | (=, [x;y]) -> x = y
      | _ -> failwith "uninterpreted predicate" in
      ([false; true], func, pred);

# holds bool_interp undefined <<forall x. (x = 0) \/ (x = 1)>>;;
- : bool = true
# holds bool_interp undefined <<forall x. (x + 1) + 1 = x>>;;
- : bool = true
let mod_interp n = 
  let func f args = 
    match (f,args) with 
    ("0",[]) -> 0 | ("1",[]) -> 1 mod n 
    | ("+",[x;y]) -> (x + y) mod n | ("*",[x;y]) -> (x * y) mod n 
    | _ -> failwith "uninterpreted function"

  and pred p args = 
    match (p,args) with 
    ("=",[x;y]) -> x = y | _ -> failwith "uninterpreted predicate" in 

  (0--(n-1),func,pred);

# holds (mod_interp 2) undefined <<forall x. (x = 0) \/ (x = 1)>>;;
  - : bool = true
# holds (mod_interp 3) undefined <<forall x. (x = 0) \/ (x = 1)>>;;
  - : bool = false
# let fm = <<forall x. ~(x = 0) ==> exists y. x * y = 1>>;;
val fm : fol formula = <<forall x. ~x = 0 ==> (exists y. x * y = 1)>>

# filter (fun n -> holds (mod_interp n) undefined fm) (1--45);
  - : int list = [1; 2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41; 43]
The Model Checker RISCAL

https://www.risc.jku.at/research/formal/software/RISCAL/
The Model Checker RISCAL

val N:N; type Num = N[N];

pred divides(m:Num,n:Num) ⇔
   ∃p:Num. m·p = n;
pred isgcd(g:Num,m:Num,n:Num) ⇔
   divides(g,m) ∧ divides(g,n) ∧
   ∀g0:Num. divides(g0,m) ∧ divides(g0,n) ⇒
   g0 ≤ g;

theorem t1() ⇔ isgcd(1,3,5);
theorem gcdbound(m:Num,n:Num) ⇔
   ∀g:Num. isgcd(g,m,n) ⇒ g ≤ m ∧ g ≤ n;
theorem gcdbound0(m:Num,n:Num) ⇔
   m ≠ 0 ∧ n ≠ 0 ⇒ ∀g:Num. isgcd(g,m,n) ⇒
   g ≤ m ∧ g ≤ n;

Using N=5.
Type checking and translation completed.
Executing t1().
Execution completed (24 ms).
Executing gcdbound(Z,Z) with all 36 inputs.
ERROR in execution of gcdbound(0,0): evaluation of
gcdbound
at line 23 in file algebra.txt:
   theorem is not true
ERROR encountered in execution (8 ms).
Executing gcdbound0(Z,Z) with all 36 inputs.
Execution completed for ALL inputs (46 ms,
   36 checked, 0 inadmissible).

First-order logic over finite domains with fixed interpretations.
The Model Checker RISCAL

Visualization of formula semantics by a “partial evaluation tree”.
Satisfiability and Validity

Let $F$ denote a first-order formula, $M = (D, I)$ a structure, $v$ a valuation.

- Formula $F$ is **satisfiable**, if there exists some structure $M$ and valuation $v$ such that $[[ F ]]_v^M = \text{true}$.
  - Example: $p(0, x)$ is satisfiable; $q(x) \land \neg q(x)$ is not.

- Structure $M$ is a **model** of formula $F$, written as $M \models F$, if for every valuation $v$, we have $[[ F ]]_v^M = \text{true}$.
  - Example: $(\mathbb{N}, [0 \mapsto \text{zero}, p \mapsto \text{less-equal}]) \models p(0, x)$

- Formula $F$ is **valid**, written as $\models F$, if every structure $M$ is a model of $F$, i.e., for every structure $M$ we have $M \models F$.
  - Example: $\models p(x) \land (p(x) \Rightarrow q(x)) \Rightarrow q(x)$
Logical Consequence and Equivalence

- Formula $F_2$ is a **logical consequence** of formula $F_1$, written as $F_1 \models F_2$, if for every structure $M$ and valuation $\nu$, the following is true:
  
  If $\llbracket F_1 \rrbracket^M_\nu = \text{true}$, then also $\llbracket F_2 \rrbracket^M_\nu = \text{true}$.

  - Example: $p(x) \land (p(x) \Rightarrow q(x)) \models q(x)$

- Formula $F$ is a **logical consequence** of formulas $F_1, \ldots, F_n$, written $F_1, \ldots, F_n \models F$, if for every $M$ and $\nu$ the following is true:
  
  If for every formula $F_i$ we have $\llbracket F_i \rrbracket^M_\nu = \text{true}$, then $\llbracket F \rrbracket^M_\nu = \text{true}$.

  - Example: $p(x), q(x) \models p(x) \land q(x)$

- Formulas $F_1$ and $F_2$ are **logically equivalent**, written as $F_1 \equiv F_2$, if and only if $F_1$ is a logical consequence of $F_2$ and vice versa, i.e., $F_1 \models F_2$ and $F_2 \models F_1$.

  - Example: $p(x) \Rightarrow q(x) \equiv \neg p(x) \lor q(x)$
Semantic Relationships

- **Satisfiability and Validity:**
  - $F$ is satisfiable, if $\neg F$ is not valid.
  - $F$ is valid, if $\neg F$ is not satisfiable.

- **Logical Consequence and Equivalence**
  - Formula $F_2$ is a logical consequence of formula $F_1$ (i.e., $F_1 \models F_2$) if and only if the formula $(F_1 \Rightarrow F_2)$ is valid.
  - Formula $F$ is a logical consequence of formulas $F_1, \ldots, F_n$ (i.e., $F_1, \ldots, F_n \models F$) if and only if the formula $(F_1 \land \ldots \land F_n \Rightarrow F)$ is valid.
  - Formula $F_1$ and formula $F_2$ are logically equivalent (i.e., $F_1 \equiv F_2$) if and only if the formula $(F_1 \Leftrightarrow F_2)$ is valid.

Logical consequence/equivalence reduced to validity of an implication/equivalence.
Assume $F \equiv F'$ and $G \equiv G'$. Then we have the following equivalences:

- $\neg F \equiv \neg F'$
- $F \land G \equiv F' \land G'$
- $F \lor G \equiv F' \lor G'$
- $F \Rightarrow G \equiv F' \Rightarrow G'$
- $F \Leftrightarrow G \equiv F' \Leftrightarrow G'$
- $\forall x \cdot F \equiv \forall x \cdot F'$
- $\exists x \cdot F \equiv \exists x \cdot F'$

Logically equivalent formulas can be substituted in any context.
Logical Equivalence: Rules

In addition to the logical equivalences for connectives in propositional logic:

\[-\forall x. F \equiv \exists x. \neg F \quad \text{(De Morgan’s Law)}\]

\[-\exists x. F \equiv \forall x. \neg F \quad \text{(De Morgan’s Law)}\]

\[\forall x. (F_1 \land F_2) \equiv (\forall x. F_1) \land (\forall x. F_2)\]

\[\exists x. (F_1 \lor F_2) \equiv (\exists x. F_1) \lor (\exists x. F_2)\]

\[\forall x. (F_1 \lor F_2) \equiv F_1 \lor (\forall x. F_2) \quad \text{if } x \notin fv(F_1)\]

\[\exists x. (F_1 \land F_2) \equiv F_1 \land (\exists x. F_2) \quad \text{if } x \notin fv(F_1)\]

For a finite domain whose values are denoted by constants \(\{c_1, \ldots, c_n\}\):

\[\forall x. F \equiv F[c_1/x] \land \ldots \land F[c_n/x]\]

\[\exists x. F \equiv F[c_1/x] \lor \ldots \lor F[c_n/x]\]
Logical Equivalence: Examples

- Push negations from the outside to the inside:

\[ \neg(\forall x. p(x) \Rightarrow \exists y. q(x, y)) \equiv \exists x. \neg(p(x) \Rightarrow \exists y. q(x, y)) \]
\[ \equiv \exists x. \neg((\neg p(x)) \vee \exists y. q(x, y)) \]
\[ \equiv \exists x. ((\neg \neg p(x)) \wedge \neg \exists y. q(x, y)) \]
\[ \equiv \exists x. (p(x) \wedge \neg \exists y. q(x, y)) \]
\[ \equiv \exists x. (p(x) \wedge \forall y. \neg q(x, y)) \]

- Reduce the scope of quantifiers:

\[ \forall x, y. (p(x) \Rightarrow q(x, y)) \equiv \forall x, y. (\neg p(x) \vee q(x, y)) \]
\[ \equiv \forall x. (\neg p(x) \vee \forall y. q(x, y)) \]
\[ \equiv \forall x. (p(x) \Rightarrow \forall y. q(x, y)) \]

- Replace quantification in a finite domain \( \{0, 1, 2\} \):

\[ \forall x. p(x) \equiv p(0) \wedge p(1) \wedge p(2) \]
Prenex Normal Form

- A formula $F$ is in prenex normal form (PNF) if it is of the following form:
  $$Q_1 x_1. \ldots Q_n x_n. M$$
  - Quantifiers $Q_1, \ldots, Q_n$.
  - Formula $M$ (the matrix) without quantifiers.

- Example:
  $$\forall x. \exists y. \forall z. P(x) \land P(y) \Rightarrow P(z)$$

- We can compute PNF by applying logical equivalences:
  - Remove quantifiers whose variable does not occur freely in body.
  - Perform the simplifications of propositional logic.
  - Compute negation normal form (“push down negations”).
  - Pull out quantifiers (renaming bound variables if necessary).

The steps can be best described by actual code.
Prenex Normal Form in OCaml

We have $Qx. F \equiv F$ if $x \notin fv(F)$.

```ocaml
let simplify1 fm =  
  match fm with  
    Forall(x,p) -> if mem x (fv p) then fm else p  
  | Exists(x,p) -> if mem x (fv p) then fm else p  
  | _ -> psimplify1 fm;;

let rec simplify fm =  
  match fm with  
    Not p -> simplify1 (Not(simplify p))  
  | And(p,q) -> simplify1 (And(simplify p,simplify q))  
  | ...  
  | Forall(x,p) -> simplify1(Forall(x,simplify p))  
  | Exists(x,p) -> simplify1(Exists(x,simplify p))  
  | _ -> fm;;

# simplify «((forall x y. P(x) \/ (P(y) \/ \ false)) ==> exists z. Q)>>;;
- : fol formula = «((forall x. P(x)) ==> Q)>>
```
Prenex Normal Form in OCaml

We have $\neg \forall x. F \equiv \exists x. \neg F$ and $\neg \exists x. F \equiv \forall x. \neg F$.

```ocaml
let rec nnf fm =
  match fm with
    And(p,q) -> And(nnf p,nnf q) | Or(p,q) -> Or(nnf p,nnf q)
| Imp(p,q) -> Or(nnf(Not p),nnf q)
| Iff(p,q) -> Or(And(nnf p,nnf q),And(nnf(Not p),nnf(Not q)))
| Not(Not p) -> nnf p
| Not(And(p,q)) -> Or(nnf(Not p),nnf(Not q)) | Not(Or(p,q)) -> And(nnf(Not p),nnf(Not q))
| Not(Imp(p,q)) -> And(nnf p,nnf(Not q))
| Not(Iff(p,q)) -> Or(And(nnf p,nnf(Not q)),And(nnf(Not p),nnf q))
| Forall(x,p) -> Forall(x,nnf p) | Exists(x,p) -> Exists(x,nnf p)
| Not(Forall(x,p)) -> Exists(x,nnf(Not p)) | Not(Exists(x,p)) -> Forall(x,nnf(Not p))
| _ -> fm;;
```

# nnf <<(forall x. P(x)) ==> ((exists y. Q(y)) <=> exists z. P(z) /\ Q(z))>>;;
- : fol formula =
<<(exists x. ~P(x))\/ (exists y. Q(y))\/ (exists z. P(z)\/ Q(z))\/ (forall y. ~Q(y))\/ (forall z. ~P(z)\/ ~Q(z))>>
Prenex Normal Form in OCaml

We have (for instance) $F_1 \lor (\forall x. F_2) \equiv \forall x. (F_1 \lor F_2)$ if $x \notin fv(F_1)$.

- Thus $F_1 \lor (\forall x. F_2) \equiv \forall y. (F_1 \lor F_2[y/x])$ if $y \notin fv(F_1) \cup fv(F_2)$.

```ocaml
let rec pullquants fm =
    match fm with
    | And(Forall(x,p),Forall(y,q)) -> pullq(true,true) fm mk_forall mk_and x y p q
    | Or(Exists(x,p),Exists(y,q)) -> pullq(true,true) fm mk_exists mk_or x y p q
    | And(Forall(x,p),q) -> pullq(true,false) fm mk_forall mk_and x x p q
    | And(p,Forall(y,q)) -> pullq(false,true) fm mk_forall mk_and y y p q
    | Or(Forall(x,p),q) -> pullq(true,false) fm mk_forall mk_or x x p q
    | Or(p,Forall(y,q)) -> pullq(false,true) fm mk_forall mk_or y y p q
    | And(Exists(x,p),q) -> pullq(true,false) fm mk_exists mk_and x x p q
    | And(p,Exists(y,q)) -> pullq(false,true) fm mk_exists mk_and y y p q
    | Or(Exists(x,p),q) -> pullq(true,false) fm mk_exists mk_or x x p q
    | Or(p,Exists(y,q)) -> pullq(false,true) fm mk_exists mk_or y y p q
    | _ -> fm

and ...
```
and pullq(l,r) fm quant op x y p q =
  let z = variant x (fv fm) in
  let p’ = if l then subst (x |=> Var z) p else p
  and q’ = if r then subst (y |=> Var z) q else q in
  quant z (pullquants(op p’ q’));;

let rec prenex fm =
  match fm with
  | Forall(x,p) -> Forall(x,prenex p)
  | Exists(x,p) -> Exists(x,prenex p)
  | And(p,q) -> pullquants(And(prenex p,prenex q))
  | Or(p,q) -> pullquants(Or(prenex p,prenex q))
  | _ -> fm;;

let pnf fm = prenex(nnf(simplify fm));;

# pnf <<(forall x. P(x) \ R(y)) ==> exists y z. Q(y) \ ~(exists z. P(z) \ Q(z))>>;;
- : fol formula = <<exists x. forall z. ~P(x) \ ~R(y) \ Q(x) \ ~P(z) \ ~Q(z)>>
Skolem Normal Form

- A formula is in Skolem normal form (SNF) if it is in prenex normal form and only contains universal quantifiers.
  - But how to remove the existential quantifiers?
- **Theorem (Skolemization):** Let $F$ be a formula with free variables $x_1, \ldots, x_n, y$. Let $f$ be an $n$-ary function symbol that does not occur in $F$. Then
  \[ \forall x_1, \ldots, x_n. \exists y. F \text{ is satisfiable if and only if } \forall x_1, \ldots, x_n. F[f(x_1, \ldots, x_n)/y] \text{ is.} \]
  - Skolem function $f$ ($n = 0$: Skolem constant $c$), substitution $F[t/x]$ of $t$ for $x$ in $F$.
  - Proof sketch: First, let $(D, I)$ and $v$ satisfy $\forall x_1, \ldots, x_n. \exists y. F$. Then for all $d_1, \ldots, d_n \in D$ there exists $d \in D$ such that $v[x_1 \mapsto d_1, \ldots, x_n \mapsto d_n, y \mapsto d]$ satisfies $F$. Thus there exists
    \[ f_D(d_1, \ldots, d_n): D^n \to D \text{ such that for all } d_1, \ldots, d_n \in D \text{ structure } (D, I) \text{ and valuation} \]
    \[ v[x_1 \mapsto d_1, \ldots, x_n \mapsto d_n, y \mapsto f_D(d_1, \ldots, d_n)] \text{ satisfy } F. \]
    Thus $\forall x_1, \ldots, x_n. F[f(x_1, \ldots, x_n)/y]$ is satisfied by structure $(D, I')$ and valuation $v$ where $I'$ is identical to $I$ except that $I'(f) := f_D$.
  
  Second, let $(D, I)$ and $v$ satisfy $\forall x_1, \ldots, x_n. F[f(x_1, \ldots, x_n)/y]$. Then, for $d_1, \ldots, d_n \in D$, $(D, I)$ and $v[x_1 \mapsto d_1, \ldots, x_n \mapsto d_n, y \mapsto I(f)(d_1, \ldots, d_n)]$ satisfy $F$. Thus $(D, I)$ and $v$ satisfy $\forall x_1, \ldots, x_n. \exists y. F$.

We can construct an equisatisfiable formula without existential quantifiers.
Skolem Normal Form in OCaml

```ocaml
let rec skolem fm fns =  
  match fm with
  | Exists(y,p) -> let xs = fv(fm) in
    let f = variant (if xs = [] then "c_"^y else "f_"^y) fns in
    let fx = Fn(f,map (fun x -> Var x) xs) in
    skolem (subst (y |=> fx) p) (f:@fns) in
  | Forall(x,p) -> let p’,fns’ = skolem p fns in Forall(x,p’),fns’ in
  | And(p,q) -> skolem2 (fun (p,q) -> And(p,q)) (p,q) fns in
  | Or(p,q) -> skolem2 (fun (p,q) -> Or(p,q)) (p,q) fns in
  | _ -> fm,fns
  and skolem2 cons (p,q) fns =  
    let p’,fns’ = skolem p fns in let q’,fns” = skolem q fns’ in
    cons(p’,q’),fns” in

let askolemize fm = fst(skolem (nnf(simplify fm)) (map fst (functions fm)));
let rec specialize fm = match fm with Forall(x,p) -> specialize p | _ -> fm;
let skolemize fm = specialize(pnf(askolemize fm));;

# pnf(askolemize <<forall x. P(x) ==> (exists y z. Q(x,y,z)) /
  (exists y z. R(y,z))>>);
- : fol formula = <<forall x. ~P(x) /
  Q(x,f_y(x),f_z(x)) /
  R(c_y,c_z)>>