2. First-Order Logic

Huixing Fang

School of Information Engineering
Yangzhou University

Outline

1 Syntax

Function

An n-ary function f takes n terms as arguments. We represent generic FOL functions by symbols f, g, h, f_1, f_2, etc. A constant can also be viewed as a 0-ary function.

Example 1

The following are all terms:

- a, a constant (or 0-ary function);
- x, a variable;
- $f(a)$, a unary function f applied to a constant;
- $g(x, b)$, a binary function g applied to a variable x and a constant b;
- $f(g(x, f(b)))$.

1 Syntax

Predicate

The propositional variables of PL are generalized to **predicates**. An n-ary predicate takes n terms as arguments. An FOL propositional variable is a 0-ary predicate.

Atom & Literal

An **atom** is $\top$, $\bot$, or an n-ary predicate applied to n terms. A **literal** is an atom or its negation.

Example 2

The following are all literals:

1. P, a propositional variable (or 0-ary predicate);
2. $p(f(x), g(x, f(x)))$, a binary predicate applied to two terms;
3. $\neg p(f(x), g(x, f(x)))$.

FOL formula

An FOL formula may be :

1. a literal;
2. application of a logical connective ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$) to a formula or formulae;
3. application of a quantifier to a formula
   - existential quantifier $\exists$. The formula $\exists x.\ F[x]$, read "there exists an x such that $F[x]$";
   - universal quantifier $\forall$. The formula $\forall x.\ F[x]$, read "for all x, F[x]".

Quantified variable & Scope

In $\forall x.\ F[x]$ (or $\exists x.\ F[x]$), x is the **quantifier vaiable**, and $F[x]$ is the **scope** of the quantifier $\forall x$ (or $\exists x$). (the scope of the quantified variable x itself)

Example 3

In

$$\forall x.\ \underbrace{p(f(x), x) \to (\exists y.\ \underbrace{p(f(g(x,y)), g(x,y))}_{G}) \wedge q(x, f(x))}_{F}$$

the scope of x is F, and the scope of y is G. This formula is read: "for all x, if $p(f(x), x)$ then there exists a y such that $p(f(g(x,y)), g(x,y))$ and $q(x, f(x))$".

Bound variable

A variable is **bound** in formula $F[x]$ if there is an occurrence of x in the scope of a binding quantifier $\forall x$ or $\exists x$. Denote by $\texttt{bound}(F)$ the set of bound variables of a formula F.

Free variable

A variable is **free** in formula $F[x]$ if there is an occurrence of x that is not bound by any quantifier. Denote by $\texttt{free}(F)$ the set of free variables of a formula F.

Is it possible that $\texttt{free}(F) \cap \texttt{bound}(F) \neq \emptyset$*?*

Example 4

$$F : \forall x.\ p(f(x), y) \to \forall y.\ p(f(x), y),$$

x only occurs bound, while y appears both free (in the antecedent) and bound (in the consequent). Thus, $\texttt{free}(F) = \{y\}$ and $\texttt{bound}(F) = \{x, y\}$.

Closed formula

A formula F is **closed** if it does not contain any free variables.

Closure

If $\texttt{free}(F) = \{x_1, ..., x_n\}$, then its **universal closure** is

$$\forall x_1.\ ...\forall x_n.\ F \text{ or } \forall * .\ F,$$

and **existential closure** is

$$\exists x_1.\ ...\forall x_n.\ F \text{ or } \exists * .\ F.$$

Subformulae

The subformulae of an FOL formula are defined according to an extension of the PL definition of subformula:

- the only subformula of $p(t_1, ..., t_n)$, where the t_i are terms, is $p(t_1, ..., t_n)$;
- the subformulae of $\neg F$ are $\neg F$ and the subformulae of F;
- the subformulae of $F_1 \wedge F_2$, $F_1 \vee F_2$, $F_1 \rightarrow F_2$, $F_1 \leftrightarrow F_2$ are the formula itself and the subformulae of F_1 and F_2;
- the subformulae of $\exists x.\ F$ and $\forall x.\ F$ are the formula itself and the subformulae of F.

The **strict subformulae** of a formula excludes the formula itself.

Subterms

The subterms of an FOL term are defined as follows:

- the only subterm of constant a or variable x is a or x itself, respectively;
- and the subterms of $f(t_1, ..., t_n)$ are the term itself and the subterms of $t_1, ..., t_n$.

The **strict subterms** of a term excludes the term itself.

Example 5

In

$$F : \forall x.\ p(f(x), y) \to \forall y.\ p(f(x), y),$$

the subformulae of F are

$$F,\ \ p(f(x), y) \to \forall y.\ p(f(x), y),\ \ \forall y.\ p(f(x), y),\ \ p(f(x), y).$$

The subterms of $g(f(x), f(h(f(x))))$ are

$$g(f(x), f(h(f(x)))),\ \ f(x),\ \ f(h(f(x))),\ \ h(f(x)),\ \ x.$$

Translations of English sentences into FOL:

1. Every dog has its day.
   $\forall x.\ dog(x) \to \exists y.\ day(y) \wedge itsDay(x, y);$
2. Some dogs have more days than others.
   $\exists x, y.\ dog(x) \wedge dog(y) \wedge \#days(x) > \#days(y)$
3. All cats have more days than dogs.
   $\forall x, y.\ dog(x) \wedge cat(y) \to \#days(y) > \#days(x)$
4. Fido is a dog. Furrball is a cat. Fido has fewer days than does Furrball.
   $dog(Fido) \wedge cat(Furrball) \wedge \#days(Fido) < \#days(Furrball)$
5. Fermat's Last Theorem.
   $$\begin{aligned} &\forall n.\ integer(n) \wedge n > 2 \\ &\quad \to \\ &\quad \forall x, y, z.\ integer(x) \wedge integer(y) \wedge integer(z) \wedge x > 0 \wedge y > 0 \wedge z > 0 \\ &\qquad \to \\ &\quad x^n + y^n \neq z^n \end{aligned}$$

Outline

- Formulae of FOL evaluate to the truth values **true** and **false** as in PL.
- Terms of FOL formulae evaluate to values from a specified domain.
- We extend the concept of interpretations to this more complex setting and then define the semantics of FOL in terms of interpretations.

FOL interpretation $\mathcal{I}$

- The **domain** $D_{\mathcal{I}}$ of $\mathcal{I}$: a nonempty set of values or objects, such as integers, real numbers, dogs, people, or merely abstract objects;
- $|D_{\mathcal{I}}|$ denotes the **cardinality** or size, of $D_{\mathcal{I}}$.
- The **assignment** $\alpha\mathcal{I}$ maps constant, variable, function, and predicate symbols to elements, functions, and predicates over $D_{\mathcal{I}}$;
- An **interpretation** $\mathcal{I} : (D_{\mathcal{I}}, \alpha\mathcal{I})$ is a pair consisting of a domain and an assignment.

Assignment $\alpha\mathcal{I}$

- Each **variable** symbol x is assigned a valued $x_\mathcal{I}$ from $D_\mathcal{I}$;
- Each $n - ary$ **function** symbol f is assigned aj n-ary function
$$f_\mathcal{I} : D_\mathcal{I}^n \to D_\mathcal{I}$$
that maps n elements of $D_\mathcal{I}$ to an element of $D_\mathcal{I}$;
- Each n-ary **predicate** symbol p is assigned an n-ary predicate
$$p_\mathcal{I} : D_\mathcal{I}^n \to \{\texttt{true}, \ \texttt{false}\}$$
that maps n elements of $D_\mathcal{I}$ to a truth value;
- Each **constant** (0-ary function symbol) is assigned a value from $D_\mathcal{I}$;
- Each **propositional variable** (0-ary predicate symbol) is assigned a truth value.

Example 6

The formula

$$F : x + y > z \rightarrow y > z - x$$

contains the binary function symbols $+$ and $-$, the binary predicate symbol $>$, and the variables x, y, and z. The domain is the integers, $\mathbb{Z}$:

$$D_{\mathcal{I}} = \mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}.$$

We thus have interpretation $\mathcal{I} : (\mathbb{Z}, \alpha_{\mathcal{I}})$, where:

$$\alpha_{\mathcal{I}} : \{+ \mapsto +_{\mathbb{Z}},\ - \mapsto -_{\mathbb{Z}},\ \ > \mapsto >_{\mathbb{Z}},\ x \mapsto 10,\ y \mapsto 8,\ z \mapsto 17,\ ...\}$$

The elision (...) reminds us that, as always, $\alpha_{\mathcal{I}}$ provides values for the countably infinitely many other constant, function, and predicate symbols.

- Given an FOL formula F and interpretation $\mathcal{I} : (D_{\mathcal{I}}, \alpha_{\mathcal{I}})$, we want to compute if F evaluates to **true** (or **false**) under interpretation $\mathcal{I}$, $\mathcal{I} \models F$ (or $\mathcal{I} \not\models F$).

Semantics

- truth symbols: $\mathcal{I} \models \top, \quad \mathcal{I} \not\models \bot$;
- $\alpha_{\mathcal{I}}$ gives meaning $\alpha_{\mathcal{I}}[x]$, $\alpha_{\mathcal{I}}[c]$, and $\alpha_{\mathcal{I}}[f]$ to variables x, constants c, and functions f;
- $\alpha_{\mathcal{I}}[f(t_1, ..., t_n)] = \alpha_{\mathcal{I}}[f](\alpha_{\mathcal{I}}[t_1], ..., \alpha_{\mathcal{I}}[t_n])$;
- $\alpha_{\mathcal{I}}[p(t_1, ..., t_n)] = \alpha_{\mathcal{I}}[p](\alpha_{\mathcal{I}}[t_1], ..., \alpha_{\mathcal{I}}[t_n])$;
- $\mathcal{I} \models p((t_1, ..., t_n)$ iff $\alpha_{\mathcal{I}}[p(t_1, ..., t_n)] = \mathtt{true}$;
- The logical connectives are handled in FOL in precisely the same way as in PL.

Example 7

Recall the formula

$$F : x + y > z \to y > z - x$$

the interpretation $\mathcal{I} : (\mathbb{Z}, \alpha_{\mathcal{I}})$, where

$$\alpha_{\mathcal{I}} : \{+ \mapsto +_{\mathbb{Z}},\ - \mapsto -_{\mathbb{Z}},\ > \mapsto >_{\mathbb{Z}},\ x \mapsto 10,\ y \mapsto 8,\ z \mapsto 17\}.$$

Compute the truth value of F under $\mathcal{I}$ as follows:

1. $\mathcal{I} \models x + y > z$	since $\alpha_{\mathcal{I}}[x + y > z] = 10 + 8 > 17$
2. $\mathcal{I} \models y > z - x$	since $\alpha_{\mathcal{I}}[y > z - x] = 8 > 17 - 10$
3. $\mathcal{I} \models F$	by 1, 2, and the semantics of $\to$

x-variant

An x-variant of an interpretation $\mathcal{I} : (\mathbb{Z}, \alpha_{\mathcal{I}})$ as an interpretation $\mathcal{J} : (\mathbb{Z}, \alpha_{\mathcal{J}})$ such that

- $D_{\mathcal{I}} = D_{\mathcal{J}}$;
- and $\alpha_{\mathcal{I}}[y] = \alpha_{\mathcal{I}}[y]$ for all constant, free variable, function, and predicate symbols y, except possibly x.

Denote by $\mathcal{J} : \mathcal{I} \triangleleft \{x \mapsto v\}$ the x-variant of $\mathcal{I}$ in which $\alpha_{\mathcal{J}}[x] = v$ for some $v \in D_{\mathcal{I}}$.

Semantics

For quntifiers,

$$\begin{array}{ll} \mathcal{I} \models \forall x.\ F & \text{iff for all } v \in D_{\mathcal{I}}, \mathcal{I} \triangleleft \{x \mapsto v\} \models F \\ \mathcal{I} \models \exists x.\ F & \text{there exists } v \in D_{\mathcal{I}}, \text{ such that } \mathcal{I} \triangleleft \{x \mapsto v\} \models F \end{array}$$

Example 8

Consider the formula

$$F : \exists x.\ f(x) = g(x)$$

and the interpretation $\mathcal{I} : (D : \{\circ, \bullet\}, \alpha_{\mathcal{I}})$ in which

$$\alpha_{\mathcal{I}} : \{f(\circ) \mapsto \circ,\ f(\bullet) \mapsto \bullet,\ g(\circ) \mapsto \bullet,\ g(\bullet) \mapsto \circ\}.$$

Compute the truth value of F under $\mathcal{I}$ as follows:

1.	$\mathcal{I} \triangleleft \{x \mapsto v\} \not\models f(x) = g(x)$	for $v \in D$
2.	$\mathcal{I} \not\models \exists x.\ f(x) = g(x)$	since $v \in D$ is arbitrary

Outline

3 Satisfiability and Validity

1. Formula F is said to be **satisfiable** iff there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models F$;
2. Formula F is said to be **valid** iff for all interpretations $\mathcal{I}$, $\mathcal{I} \models F$;
3. Satisfiability and validity are **dual**: F is valid iff $\neg F$ is unsatisfiable.

For arguing the validity of FOL formulae, we extend the semantic argument method from PL to FOL.

Extended Semantic Argument Method

According to the semantics of universal quantification, from $\mathcal{I} \models \forall x.\ F$, deduce $\mathcal{I} \triangleleft \{x \mapsto v\} \models F$ for any $v \in D_{\mathcal{I}}$.

$$\frac{\mathcal{I} \models \forall x.\ F}{\mathcal{I} \triangleleft \{x \mapsto v\} \models F} \text{ for any } v \in D_{\mathcal{I}}$$

In practice, we usually apply this rule using a domain element v that was introduced earlier in the proof.

Extended Semantic Argument Method

Similarly, from the semantics of existential quantification, from $\mathcal{I} \not\models \exists x.\ F$, deduce $\mathcal{I} \triangleleft \{x \mapsto v\} \not\models F$ for any $v \in D_{\mathcal{I}}$. used in the proof.

$$\frac{\mathcal{I} \not\models \exists x.\ F}{\mathcal{I} \triangleleft \{x \mapsto v\} \not\models F} \text{ for any } v \in D_{\mathcal{I}}$$

Again, we usually apply this rule using a domain element v that was introduced earlier in the proof.

Extended Semantic Argument Method

According to the semantics of existential quantification, from $\mathcal{I} \models \exists x.\ F$, deduce $\mathcal{I} \triangleleft \{x \mapsto v\} \models F$ for some $v \in D_{\mathcal{I}}$ that has **not** been previously used in the proof.

$$\frac{\mathcal{I} \models \exists x.\ F}{\mathcal{I} \triangleleft \{x \mapsto v\} \models F} \text{ for a } \textbf{fresh } v \in D_{\mathcal{I}}$$

Extended Semantic Argument Method

Similarly, from the semantics of universal quantification, from $\mathcal{I} \models \forall x.\ F$, deduce $\mathcal{I} \triangleleft \{x \mapsto v\} \not\models F$ for some $v \in D_{\mathcal{I}}$ that has **not** been previously used in the proof.

$$\frac{\mathcal{I} \not\models \forall x.\ F}{\mathcal{I} \triangleleft \{x \mapsto v\} \not\models F} \text{ for a } \textbf{fresh } v \in D_{\mathcal{I}}$$

Extended Semantic Argument Method

A contradiction exists if two variants of the original interpretation $\mathcal{I}$ disagree on the truth value of an n-ary predicate p for a given tuple of domain values.

$$\frac{\begin{array}{c} J : \mathcal{I} \lhd \dots \models p(s_1, \dots, s_n) \\ K : \mathcal{I} \lhd \dots \not\models p(t_1, \dots, t_n) \end{array}}{\mathcal{I} \models \bot} \text{ for } i \in \{1, \dots, n\},\ \alpha_J[s_i] = \alpha_K[t_i]$$

Example 9

We prove that

$$F : (\forall x.\ p(x)) \rightarrow (\forall y.\ p(y))$$

is valid. Suppose not; and $\mathcal{I} \not\models F$:

1.	$\mathcal{I} \not\models F$	assumption
2.	$\mathcal{I} \models \forall x.\ p(x)$	1 and semantics of $\rightarrow$
3.	$\mathcal{I} \not\models \forall y.\ p(y)$	1 and semantics of $\rightarrow$
4.	$\mathcal{I} \triangleleft \{y \mapsto v\} \not\models p(y)$	3 and semantics of $\forall$, for some $v \in D_{\mathcal{I}}$
5.	$\mathcal{I} \triangleleft \{x \mapsto v\} \models p(x)$	2 and semantics of $\forall$

under $\mathcal{I}$, $p(v)$ is **false** by 4 and **true** by 5. Thus, F is valid.

Example 10

Consider the following relation between universal and existential quantification:

$$F : (\forall x.\ p(x)) \leftrightarrow (\neg \exists x.\ \neg p(x))\ .$$

Suppose not. Then there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \not\models F$.
In the first case (forward $\rightarrow$),

1.	$\mathcal{I} \models \forall x.\ p(x)$	assumption
2.	$\mathcal{I} \not\models \neg \exists x.\ \neg p(x)$	assumption
3.	$\mathcal{I} \models \exists x.\ \neg p(x)$	2 and $\neg$
4.	$\mathcal{I} \triangleleft \{x \mapsto v\} \models \neg p(x)$	3 and $\exists$, for some $v \in D_{\mathcal{I}}$
5.	$\mathcal{I} \triangleleft \{x \mapsto v\} \models p(x)$	1 and $\forall$

Continue Example 10. For the second case (backward $\leftarrow$),

1.	$\mathcal{I} \not\models \forall x.\ p(x)$	assumption
2.	$\mathcal{I} \models \neg\exists x.\ \neg p(x)$	assumption
3.	$\mathcal{I} \triangleleft \{x \mapsto v\} \not\models p(x)$	1 and $\forall$, for some $v \in D_{\mathcal{I}}$
4.	$\mathcal{I} \not\models \exists x.\ \neg p(x)$	2 and $\neg$
5.	$\mathcal{I} \triangleleft \{x \mapsto v\} \not\models \neg p(x)$	4 and $\exists$
6.	$\mathcal{I} \triangleleft \{x \mapsto v\} \models p(x)$	5 and $\neg$

Both cases end in contradictions for arbitrary interpretation $\mathcal{I}$, F is valid.

Example 11

To prove that

$$F : p(a) \rightarrow \exists x.\ p(x)$$

is valid, assume otherwise and derive a contradiction.

1.	$\mathcal{I} \not\models F$	assumption
2.	$\mathcal{I} \models p(a)$	1 and $\rightarrow$
3.	$\mathcal{I} \not\models \exists x.\ p(x)$	1 and $\rightarrow$
4.	$\mathcal{I} \triangleleft \{x \mapsto \alpha_{\mathcal{I}}[a]\} \not\models p(x)$	3 and $\exists$
5.	$\mathcal{I} \models \bot$	2, 4

Because lines 2 and 4 are contradictory, F is valid.

Outline

Renaming

If variable x is **quantified** in F so that F has the form $F[\forall x.\ G[x]]$, then the **renaming** of x to fresh variable x' produces the formula $F[\forall x'.\ G[x']]$.

By the semantics of universal/existential quantification, the original and final formulae are equivalent.

Example 12

Renaming the bound variable x to fresh variable x' in

$$F : p(x) \wedge \forall x.q(x, y)$$

produces

$$F' : p(x) \wedge \forall x'.q(x', y)\ .$$

Substitution

A substitution is a map from FOL formulae to FOL formulae:

$$\sigma : \{F_1 \to G_1, ..., F_n \to G_n\} \ .$$

1. As in PL, $\mathtt{domain}(\sigma) = \{F_1, ..., F_n\}$ and $\mathtt{range}(\sigma) = \{G_1, ..., G_n\}$;
2. $F\sigma$: application of σ to F, replacing each occurrence of F_i in F by G_i **simultaneously**;
3. If $F_j, F_k \in \mathtt{domain}(\sigma)$, and F_k is a strict subformula of F_j, replace occurrences of F_j by G_j.

Example 13

Consider formula

$$F : (\forall x.\ p(x,y)) \to q(f(y),x)$$

and substitution

$$\sigma : \{x \mapsto g(x),\ y \mapsto f(x),\ q(f(y),x) \mapsto \exists x.\ h(x,y)\}\ .$$

Then

$$F\sigma : (\forall x.\ p(g(x),f(x))) \to \exists x.\ h(x,y)\ .$$

Example 14

Consider formula

$$F : \exists y.\ p(x,y) \wedge p(y,x)$$

and substitution

$$\sigma : \{\exists y.\ p(x,y) \mapsto p(x,a)\}\ ,$$

where a is a constant. Then $F\sigma = ?$.

Example 13

Consider formula

$$F : (\forall x.\ p(x, y)) \to q(f(y), x)$$

and substitution

$$\sigma : \{x \mapsto g(x),\ y \mapsto f(x),\ q(f(y), x) \mapsto \exists x.\ h(x, y)\} \ .$$

Then

$$F\sigma : (\forall x.\ p(g(x), f(x))) \to \exists x.\ h(x, y) \ .$$

Example 14

Consider formula

$$F : \exists y.\ p(x, y) \land p(y, x)$$

and substitution

$$\sigma : \{\exists y.\ p(x, y) \mapsto p(x, a)\} \ ,$$

where a is a constant. Then $F\sigma = ?$. F. The scope of the quantifier $\exists y$ in F is $p(x, y) \land p(y, x)$ not just $p(x, y)$.

4.1 Safe Substitution

Free Variables of Substitution

Define for a substitution σ its set of free variables:

$$V_\sigma = \bigcup_i (\mathtt{free}(F_i) \cup \mathtt{free}(G_i)) \ .$$

V_σ consists of the free variables of all formulae F_i and G_i of the domain and range of σ.

Safe Substitution

Compute the safe substitution $F\sigma$ of formula F as follows:

1. For each **quantified variable** x in F such that $x \in V\sigma$, rename x to a fresh variable to produce F';
2. Compute $F'\sigma$.

4.1 Safe Substitution

Example 15

Consider again formula

$$F : (\forall x.\ p(x,y)) \to q(f(y),x)$$

and substitution

$$\sigma : \{x \mapsto g(x),\ y \mapsto f(x),\ q(f(y),x) \mapsto \exists x.\ h(x,y)\}\ .$$

To compute the safe substitution $F\sigma$, first compute free variables

$$\begin{aligned} V\sigma &= \texttt{free}(x) \cup \texttt{free}(g(x)) \cup \texttt{free}(y) \cup \texttt{free}(f(x)) \\ &\quad \cup \texttt{free}(q(f(y),x)) \cup \texttt{free}(\exists x.\ h(x,y)) \\ &= \{x,y\} \end{aligned}$$

Then

1. As $x \in V\sigma$, after renaming, $F' : (\forall x'.\ p(x',y)) \to q(f(y),x)$;
2. $F'\sigma : (\forall x'.\ p(x',f(x))) \to \exists x.\ h(x,y)$.

4.1 Safe Substitution

Example 16

Consider formula

$$F : (\forall z.p(z, y)) \to q(f(y), x) \ ,$$

in which the quantified variable has a different name than any free variable of F or the substitution

$$\sigma : \{x \mapsto g(x),\ y \mapsto f(y),\ q(f(y), x) \mapsto \exists w.\ h(w, y)\} \ .$$

The safe substitution is the unrestricted substitution

$$F\sigma : (\forall z.\ p(z, f(y))) \to \exists w.\ h(w, y) \ .$$

Proposition 17 (Substitution of Equivalent Formulae)

Consider substitution

$$\sigma : \{F_1 \mapsto G_1, ..., F_n \mapsto G_n\}$$

such that for each i, $F_i \Leftrightarrow G_i$. Then $F \Leftrightarrow F\sigma$ when $F\sigma$ is computed as a safe substitution.

4.2 Schema Substitution

Formula Schema

A formula schema H, e.g., $(\forall x.\ F) \leftrightarrow (\neg\exists x.\ \neg F)$:

1. contains at least one **placeholder** $F_1, F_2, \ldots$;
2. may have **side conditions** that specify that certain variables do not occur free in the placeholders.

Schema Substitution

Consider a substitution σ mapping placeholders to FOL formulae. A schema substitution is an (unrestricted) application of σ to a formula schema.

A schema substitution is **legal** only if the substitution σ **obeys** the side conditions of the formula schema.

Example 18

Recall from Example 10 that

$$(\forall x.\ p(x)) \leftrightarrow (\neg\exists x.\ \neg p(x))$$

is valid. Rewrite the formula using placeholders:

$$H : (\forall x.\ F) \leftrightarrow (\neg\exists x.\ \neg F)\ .$$

H is a formula schema. The validity of

$$G : (\forall x.\ \exists y.\ q(x,y)) \leftrightarrow (\neg\exists x.\ \neg\exists y.\ q(x,y)s)$$

is derivable from H by the schema substitution $H\sigma$ (syntactically identical to G) by:

$$\sigma : \{F \mapsto \exists y.\ q(x,y)\}\ .$$

Example 19

Consider the formula schema with side condition

$$H : (\forall x.\ F) \leftrightarrow F \quad \text{provided } x \notin \mathtt{free}(F)\ .$$

If we disregard the side condition, then H is an invalid formula schema as, for example,

$$G_1 : (\forall x.\ p(x)) \leftrightarrow p(x)\ ,$$

obtained from H by schema substitution

$$\sigma : \{F \mapsto p(x)\}\ ,$$

is invalid. However, σ is disallowed by the side condition. A legal schema substituion can be:

$$\sigma : \{F \mapsto \exists y.\ p(z, y)\}\ ,$$

which obeys H's side condition.

4.2 Schema Substitution

Example 20

To prove the validity of

$$H : (\forall x.\ F) \leftrightarrow F \quad \text{provided } x \notin \texttt{free}(F)\ ,$$

consider the two directions of $\leftrightarrow$. First $(\rightarrow)$,

1.	$\mathcal{I} \models \forall x.\ F$	assumption
2.	$\mathcal{I} \not\models F$	assumption
3.	$\mathcal{I} \models F$	1, $\forall$, since$x \notin \texttt{free}(F)$
4.	$\mathcal{I} \models \bot$	2, 3

Second $(\leftarrow)$, similar to the first case. Thus, H is a valid formula schema.

4.2 Schema Substitution

Proposition 21 (Formula Schema)

If H is a valid formula schema and σ is a substitution obeying H's side conditions, then $H\sigma$ is also valid.

The valid PL formula

$$(P \to Q) \leftrightarrow (\neg P \vee Q)$$

can be treated as a valid formula schema:

$$(F_1 \to F_2) \leftrightarrow (\neg F_1 \vee F_2) \ .$$

In general, valid propositional templates are valid formulae schemata.

Outline

5 Normal Forms

- The normal forms of PL extend to FOL;
- An FOL formula F can be transformed into negation normal form (NNF) by using the procedure in PL augmented with these two equivalences:

$$\neg\forall x.\ F[x] \Leftrightarrow \exists x. \neg F[x]\ ,$$
$$\neg\exists x.\ F[x] \Leftrightarrow \forall x. \neg F[x]\ .$$

Example 22

Find a formula in NNF that is equivalent to

$$G : \forall x.\ (\exists y.\ p(x,y) \land p(x,z)) \rightarrow \exists w.\ p(x,w)\ .$$

Each formula below is equivalent to G and is obtained from the previous one through an application of an equivalence.

1. $\forall x.\ (\exists y.\ p(x,y) \land p(x,z)) \rightarrow \exists w.\ p(x,w)$

   $\downharpoonleft\ F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \lor F_2$

2. $\forall x.\ \neg(\exists y.\ p(x,y) \land p(x,z)) \lor \exists w.\ p(x,w)$

   $\downharpoonleft\ \neg\exists x.\ F[x] \Leftrightarrow \forall x.\ \neg F[x]$

3. $\forall x.\ (\forall y.\ \neg(p(x,y) \land p(x,z))) \lor \exists w.\ p(x,w)$

   $\downharpoonleft\ \neg(F_1 \land F_2) \Leftrightarrow \neg F_1 \lor \neg F_2$

4. $\forall x.\ (\forall y.\ \neg p(x,y) \lor \neg p(x,z)) \lor \exists w.\ p(x,w)$

Prenex Normal Form (PNF)

A formula is in prenex normal form (PNF) if all of its quantifiers appear at the beginning of the formula:

$$Q_1 x_1. \ldots Q_n x_n.\ F[x_1, \ldots, x_n]\ ,$$

where $Q_i \in \{\forall, \exists\}$ and F is quantifier-free.

Example 23

FOL formula in PNF:

$$\forall x.\ \exists y.\ \forall z.\ p(x, y) \land q(y, z)$$

- An FOL formula is in CNF (or DNF) if it is in PNF and its main quantifier-free subformula is in CNF (or DNF).

Translation of FOL Formula into PNF

To compute an equivalent PNF F' of FOL formula F,

1. Convert F into NNF formula F_1.
2. When multiple quantified variables have the same name, rename them to fresh variables, resulting in F_2.
3. Remove all quantifiers from F_2 to produce quantifier-free formula F_3.
4. Add the quantifiers before F_3,

$$F_4 : Q_1 x_1. \ldots Q_n x_n. \; F_3 \; ,$$

where the Q_i are the quantifiers such that if Q_j is in the scope of Q_i in F_1, then $i < j$.

Example 24

Find a PNF equivalent of

$$F : \forall x.\ \neg(\exists y.\ p(x,y) \wedge p(x,z)) \vee \exists y.\ p(x,y).$$

1. Write F in NNF:

$$F_1 : \forall x.\ (\forall y.\ \neg p(x,y) \vee \neg p(x,z)) \vee \exists y.\ p(x,y)\ .$$

2. Rename quantified variables:

$$F_2 : \forall x.\ (\forall y.\ \neg p(x,y) \vee \neg p(x,z)) \vee \exists w.\ p(x,w)\ .$$

3. Remove all quantifiers to produce quantifier-free formula

$$F_3 : \neg p(x,y) \vee \neg p(x,z) \vee p(x,w)\ .$$

4. Add the quantifiers before F_3:

$$F_4 : \forall x.\ \forall y.\ \exists w.\ \neg p(x,y) \vee \neg p(x,z) \vee p(x,w)\ .$$

Outline

Satisfiability as a Language

Let L_{PL} be the set of all satisfiable formulae. That is, the word $w \in L_{PL}$ iff

1. w is a syntactically well-formed formulae;
2. and when w is viewed as a PL formula F, F is satisfiable.

Then the formal decision problem (satisfiability of formulae) is:

given a word w, is $w \in L_{PL}$?

Satisfiability of FOL formulae can be similarly formalized as a language question: given a word w, is $w \in L_{FOL}$?

Decidable

A language L is **decidable** if there exists a procedure that, given a word w,

1. eventually **halts**;
2. and **answers** *yes* if $w \in L$;
3. and **answers** *no* if $w \notin L$.

Other terms for "decidable" are **recursive** and **Turing-decidable**.

- A procedure for a decidable language is called an **algorithm**
- Satisfiability of PL formulae is decidable: the truth-table method is a decision procedure
- A language is **undecidable** if it is not decidable
- Church and Turing showed that L_{FOL} is undecidable

Semi-Decidable

A language L is **semi-decidable** if there exists a procedure that, given a word w,

1. halts and **answers** *yes* **iff** $w \in L$;
2. halts and **answers** no **if** $w \notin L$;
3. or **does not halt if** $w \notin L$.

Other terms for "semi-decidable" are **partially decidable**, **recursively enumerable**, and **Turing-recognizable**.

- Unlike a decidable language, the procedure is only guaranteed to halt if $w \in L$
- The terms "Turing-decidable" and "Turing-recognizable" arise from Alan Turing's classic formalization of procedures as Turing machines

Polynomial-Time Decidable

A language L is **polynomial-time decidable**, or in **PTIME** (also, in **P**), if there exists a procedure that, given w,

1. answers *yes* when $w \in L$;
2. answers *no* when $w \notin L$;
3. and halts in a number of steps that is at most **proportionate** to some **polynomial** of the **size** of w.

- Determining if the word w is a well-formed FOL formula is polynomial-time decidable (standard parsing methods).

Nondeterministic-Polynomial-Time Decidable

A language L is **nondeterministic-polynomial-time decidable**, or in **NPTIME** (also, in **NP**), if there exists a **nondeterministic** procedure that, given w,

1. **guesses** a **witness** W to the fact that $w \in L$ that is at most proportionate in size to some **polynomial** of the size of w;
2. **checks** in time at most proportionate to some **polynomial** of the size of w that W really is a witness to $w \in L$;
3. and **answers** *yes* if the check succeeds and *no* otherwise.

L_{PL} is in NP, nondeterministic procedure for deciding satisfiable:

1. parse the input w as formula F (return no if w is not a well-formed PL formula);
2. guess an interpretation I, which is linear in the size of w;
3. check that $I \models F$.

co-NP

A language L is in **co-NP** if its **complement** language L is in NP.

- Unsatisfiability of PL formulae is in co-NP as satisfiability is in NP
- It is not known if unsatisfiability of PL formulae is in NP
- A satisfiable PL formula has a polynomial size witness of its satisfiability

NP-hard

A language L is **NP-hard** if every instance $v \in L'$ of every other NP decidable language L' can be **reduced** to deciding an instance $w^v_{L'} \in L$. Moreover, the size of $w^v_{L'}$ must be at most proportionate to some polynomial of the size of v.

NP-complete

A language L is **NP-complete** if it is in NP and is NP-hard.

- L_{PL} is NP-complete. L_{PL}(also called **SAT**) was the first language shown to be NP-complete
- The Cook-Levin theorem shows that all NP-languages L can be reduced to L_{PL}

6.2 Complexity

Standard Notation

1. $O(f(n))$: the set of all functions of at most order $f(n)$,
a function $g(n)$ is of at most order $f(n)$ if there exist a scalar $c \geq 0$ and an integer $n_0 \geq 0$ such that
$$\forall n \geq n_0.\ g(n) \leq cf(n)\ .$$

2. $\Omega(f(n))$: the set of all functions of at least order $f(n)$,
a function $g(n)$ is of at least order $f(n)$ if there exist a scalar $c \geq 0$ and an integer $n_0 \geq 0$ such that
$$\forall n \geq n_0.\ g(n) \geq cf(n)\ .$$

3. $\Theta(f(n))$: the set of all functions of precisely order $f(n)$.
$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

Example 25

1. $3n^2 + n \in O(n^2)$;
2. $3n^2 + n \in \Omega(n^2)$;
3. $3n^2 + n \in \Theta(n^2)$;
4. $\frac{1}{99}n^2 + n \in \Omega(n^2)$;
5. $3n^2 + n \in O(2^n)$;
6. $3n^2 + n \in \Omega(n)$;
7. $3n^2 + n \notin \Omega(2^n)$;
8. $3n^2 + n \notin \Theta(2^n)$;
9. $2^n \in \Omega(n^3)$;
10. $2^n \notin O(n^3)$.

Complexity of Decision Problem

A decision problem D has time complexity:

1. $O(f(n))$
   if there exists an algorithm P for D and a function $g(n) \in O(f(n))$ such that P runs in time at most $g(n)$ on input of size n
2. $\Omega(f(n))$
   if there exists a function $g(n) \in \Omega(f(n))$ such that all algorithms P for D runs in time at least $g(n)$ on input of size n.
3. $\Theta(f(n))$
   if it has time complexities $\Omega(f(n))$ and $O(f(n))$.

Outline

7 Sound and Complete

Semantic Argument Method: To show FOL formula F is valid, assume $\mathcal{I} \not\models F$, and derive a contradiction $\mathcal{I} \models \perp$ in all branches.

Theorem 26 (Sound)

If every branch of a semantic argument proof of $\mathcal{I} \not\models F$ closes (i.e., reaches $\mathcal{I} \models \perp$), then F is valid

Theorem 27 (Complete)

Each valid formula F has a semantic argument proof in which every branch is closed (i.e., reaches $\mathcal{I} \models \perp$).

7.1 Semantic Tableaux

Example 28

Consider the formula $F : p \wedge (\neg q \vee \neg p)$. The semantic tableauf of F is

$$
\begin{tikzcd}[column sep=small]
& p \wedge (\neg q \vee \neg p) \arrow[d] & \\
& p, \neg q \vee \neg p \arrow[dl] \arrow[dr] & \\
p, \neg q & & p, \neg p \\
\odot & & \times
\end{tikzcd}
$$

- The initial formula labels the root of the tree, each node has one or two child
- A leaf labeled by a set of literals containing a complementary pair of literals is marked $\times$
- A leaf labeled by a set not containing a complementary pair is marked $\odot$

7.1 Semantic Tableaux

A concise presentation of the rules for creating a semantic tableau can be given if formulas are classified according to their principal operator

Classification of α-formulae and β-formulae

For α-formulae: α-formulas are conjunctive and are satisfiable only if both subformulas α_1 and α_1 are satisfied:

α	α_1	α_2
$\neg\neg A_1$	A_1	
$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \rightarrow A_2)$	A_1	$\neg A_2$
$A_1 \leftrightarrow A_2$	$A_1 \rightarrow A_2$	$A_2 \rightarrow A_1$

A concise presentation of the rules for creating a semantic tableau can be given if formulas are classified according to their principal operator

Classification of α-formulae and β-formulae

For β-formulae: β-formulas are disjunctive and are satisfied even if only one of the subformulas β_1 or β_2 is satisfiable:

β	β_1	β_2
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$B_1 \vee B_2$	B_1	B_2
$B_1 \rightarrow B_2$	$\neg B_1$	B_2
$\neg(B_1 \leftrightarrow B_2)$	$\neg(B_1 \rightarrow B_2)$	$\neg(B_2 \rightarrow B_1)$

Algorithm of Construction of a semantic tableau
Input: A formula ϕ of propositional logic
Output: A semantic tableau $\mathcal{T}$ for ϕ all of whose leaves are marked.

Initially, $\mathcal{T}$ is a tree consisting of a single root node labeled with the singleton set $\{\phi\}$. This node is not marked.

Repeat the following step as long as possible: Choose an unmarked leaf ℓ labeled with a set of formulas $U(\ell)$ and apply construction rules.

Algorithm of Construction of a semantic tableau

Construction rules:

- $U(\ell)$ is a set of literals. Mark the leaf closed $\times$ if it contains a complementary pair of literals. If not, mark the leaf open $\odot$.
- $U(\ell)$ is not a set of literals. Choose a formula in $U(\ell)$ which is not a literal. Classify the formula as an α-formula A or as a β-formula B :
  - A is an α-formula. Create a new node ℓ' as a child of ℓ and label ℓ' with:
  $$U(\ell') = (U(\ell) - \{A\}) \cup \{A_1, A_2\}.$$
  - B is an β-formula. Create a new node ℓ' and ℓ'' as children of ℓ. Label ℓ' with:
  $$U(\ell') = (U(\ell) - \{B\}) \cup \{B_1\},$$
  and label ℓ'' with:
  $$U(\ell'') = (U(\ell) - \{B\}) \cup \{B_2\}.$$

Definition 29 (Completed Tableau, Closed, Open)

A tableau whose construction has terminated is a **completed** tableau. A completed tableau is **closed** if all its leaves are marked closed. Otherwise (if some leaf is marked open), it is **open**.

Theorem 30

The construction of a tableau for any formula ϕ ***terminates****. When the construction terminates, all the leaves are marked $\times$ or $\odot$.*

- A branch can always be extended if its leaf is labeled with a set of formulas that is not a set of literals.

Theorem 31 (Soundness and Completeness)

Let $\mathcal{T}$ be a completed tableau for a formula A. A is unsatisfiable if and only if $\mathcal{T}$ is closed.

Corollary 32

Formula A is satisfiable if and only if $\mathcal{T}$ is open.

Proof: A is satisfiable iff (by definition) A is not unsatisfiable iff $\mathcal{T}$ is not closed iff (by definition) $\mathcal{T}$ is open.

Corollary 33

Formula A is valid if and only if the tableau for $\neg A$ closes.

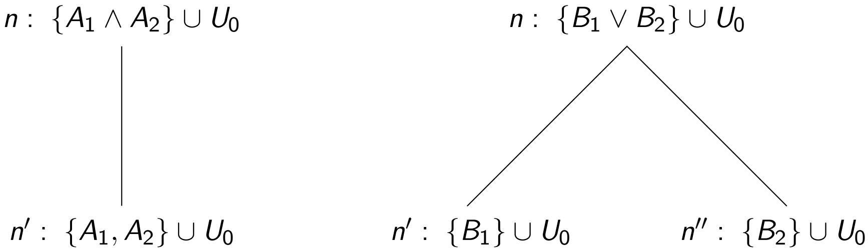
Proof: A is valid iff $\neg A$ is unsatisfiable iff the tableau for $\neg A$ closes.

- More general theorem: if $\mathcal{T}_n$, the subtree rooted at node n of $\mathcal{T}$, closes then the **set of formulas $U(n)$ labeling n** is unsatisfiable
- For simplicity, use $A_1 \wedge A_2$ and $B_1 \vee B_2$ as representatives of the classes of α- and β-formulas

Proof: The proof is by induction on the height h_n of the node n in $\mathcal{T}_n$

- Base Case: $h_n = 0$, and assume that T_n closes.
  $(h_n = 0) \Rightarrow n$ is a leaf $\Rightarrow U(n)$ contains complementary pair $\Rightarrow$ unsatisfiable.
- Inductive Step: let n be a node such that $h_n > 0$ in $\mathcal{T}_n$.
  Show that: $\mathcal{T}_n$ is closed $\Rightarrow U(n)$ is unsatisfiable.
  Assume: for any node m of height $h_m < h_n$, if T_m closes, then $U(m)$ is unsatisfiable.

Since $h_n > 0$, the rule for some α- or β-formula was used to create the children of n:



Two Cases:

1. $U(n) = \{A_1 \wedge A_2\} \cup U_0$ and $U(n') = \{A_1, A_2\} \cup U_0$ for some (possibly empty) set of formulas U_0
2. $U(n) = \{B_1 \vee B_2\} \cup U_0$, $U(n') = \{B_1\} \cup U_0$, and $U(n'') = \{B_2\} \cup U_0$ for some (possibly empty) set of formulas U_0

7.2.1 Proof of Soundness

First Case: Clearly, $\mathcal{T}_{n'}$ is also a closed tableau and since $h_{n'} = h_n - 1$, by the inductive hypothesis $U(n') = \{A_1, A_2\} \cup U_0$ is unsatisfiable.
Let $\mathcal{I}$ be an arbitrary interpretation.

1. $\mathcal{I} \not\models A_0$ for some formula $A_0 \in U_0$. But $U_0 \subset U(n)$ so $U(n)$ is also unsatisfiable
2. Otherwise, $\mathcal{I} \models A_0$ for all $A_0 \in U_0$ so $\mathcal{I} \not\models A_1$ or $\mathcal{I} \not\models A_2$.
   Suppose that
   $$\mathcal{I} \not\models A_1 \ .$$
   By the definition of the semantics of $\wedge$, this implies that
   $$\mathcal{I} \not\models A_1 \wedge A_2 \ .$$
   Since $A_1 \wedge A_2 \in U(n)$, $U(n)$ is unsatisfiable. A similar argument holds if $\mathcal{I} \not\models A_2$.

Second Case: Clearly, $\mathcal{T}_{n'}$ and $\mathcal{T}_{n''}$ are also closed tableaux and since $h_{n'} \leq h_n - 1$ and $h_{n''} \leq h_n - 1$, by the inductive hypothesis $U(n') = \{B_1\} \cup U_0$ and $U(n'') = \{B_2\} \cup U_0$ are both unsatisfiable. Let $\mathcal{I}$ be an arbitrary interpretation.

1. $\mathcal{I} \not\models B_0$ for some formula $B_0 \in U_0$. But $U_0 \subset U(n)$ so $U(n)$ is also unsatisfiable
2. Otherwise, $\mathcal{I} \models B_0$ for all $B_0 \in U_0$ so

$$\mathcal{I} \not\models B_1 \quad \text{since } U(n') \text{ is unsatisfiable ,}$$
$$\mathcal{I} \not\models B_2 \quad \text{since } U(n'') \text{ is unsatisfiable.}$$

By the definition of the semantics of $\vee$, this implies that

$$\mathcal{I} \not\models B_1 \vee B_2 \ .$$

Since $B_1 \vee A_2 \in U(n)$, $U(n)$ is unsatisfiable. $\square$

Completeness

The theorem to be proved is:

If A is unsatisfiable then every tableau for A closes.

Rather than prove the above, we prove the contrapositive:

*If **some** tableau for A is **open**, then A is **satisfiable**.*

Example 34

The tableau for formula $F = p \land (\neg q \lor \neg p)$ is:

$$
\begin{array}{ccc}
 & p \land (\neg q \lor \neg p) & \\
 & \downarrow & \\
 & p, \neg q \lor \neg p & \\
\swarrow & & \searrow \\
p, \neg q & & p, \neg p \\
\odot & & \times
\end{array}
$$

$\mathcal{I} : \{p \mapsto \top, q \mapsto \bot\}$ satisfies F

Implication and Contrapositive

$$P \to Q \Leftrightarrow \neg P \lor Q \Leftrightarrow \neg\neg Q \lor \neg P \Leftrightarrow \neg Q \to \neg P$$

7.2.2 Proof of Completeness

There are four steps in the proof:

1. Define a property of sets of formulas;
2. Show that the union of the formulas labeling nodes in an open branch has this property;
3. Prove that any set having this property is satisfiable;
4. Note that the formula labeling the root is in the set.

7.2.2 Proof of Completeness

Step-1: Define a property of sets of formulas;

Definition 35 (Hintikka set)

Let U be a set of formulas. U is a **Hintikka set** iff:

1. For all atoms p appearing in a formula of U, either $p \notin U$ or $\neg p \notin U$.
2. If $A \in U$ is an α-formula, then $A_1 \in U$ and $A_2 \in U$.
3. If $B \in U$ is a β-formula, then $B_1 \in U$ or $B_2 \in U$.

Example 36

We claim that $U = \{p, p \vee (\neg q \wedge \neg p)\}$ is a Hintikka set.

1. Condition (1) obviously holds since there is only one literal p in U and $\neg p \notin U$.
2. Condition (2) is vacuous.
3. For Condition (3), $B = p \vee (q \wedge \neg q) \in U$ is a β-formula and $B_1 = p \in U$.

7.2.2 Proof of Completeness

Step-2: Show that the union of the formulas labeling nodes in an open branch has this property;

Theorem 37

Let ℓ be an open leaf in a completed tableau $\mathcal{T}$. Let $U = \bigcup_i U(i)$, where i runs over the set of nodes on the branch from the root to ℓ. Then U is a Hintikka set.

Proof:

- If a literal p or $\neg p$ appears for the first time in $U(n)$ for some n, the literal will be copied into $U(k)$ for all nodes k on the branch from n to ℓ.
- This means that all literals in U appear in $U(\ell)$.
- Since the branch is open, no complementary pair of literals appears in $U(\ell)$, so Condition (1) holds for U.

7.2.2 Proof of Completeness

Continue Proof:

- Suppose that $A \in U$ is an α-formula.
- Since the tableau is completed, A was the formula selected for decomposing at some node n in the branch from the root to ℓ.
- Then $\{A_1, A_2\} \subseteq U(n') \subseteq U$, so Condition (2) holds.

- Suppose that $B \in U$ is a β-formula
- Since the tableau is completed, B was the formula selected for decomposing at some node n in the branch from the root to ℓ.
- Then either $B_1 \in U(n') \subseteq U$ or $B_2 \in U(n') \subseteq U$, so Condition (3) holds.

$\square$

Step-3: Prove that any set having this property is satisfiable;

Theorem 38 (Hintikka's Lemma)

Let U be a Hintikka set. Then U is satisfiable.

Proof: We define an interpretation and then show that the interpretation is a model of U. Let $\mathcal{P}_U$ be set of all **atoms** appearing in all formulas of U. **Define an interpretation** $\mathcal{I} : \mathcal{P}_U \to \{\top, \bot\}$ as follows:

$$\begin{array}{ll} \mathcal{I} \models p & \text{if } p \in U, \\ \mathcal{I} \not\models p & \text{if } \neg p \in U, \\ \mathcal{I} \models p & \text{if } p \notin U \text{ and } \neg p \notin U. \end{array}$$

Since U is a Hintikka set, by Condition (1), every atom in $\mathcal{P}_U$ is given exactly one value.

7.2.2 Proof of Completeness

Continue Proof: We show by structural induction that for any $A \in U$, $\mathcal{I} \models A$:

- If A is an atom p, then $\mathcal{I} \models A$ because $\mathcal{I} \models p$ since atom $p \in U$.
- If A is a negated atom $\neg p$, then since $\neg p \in U$, $\mathcal{I} \not\models p$, so $\mathcal{I} \not\models A$.
- If A is an α-formula, by Condition (2) $A_1 \in U$ and $A_2 \in U$. By the inductive hypothesis, $\mathcal{I} \models A_1$ and $\mathcal{I} \models A_2$, so $\mathcal{I} \models A$ by definition of the conjunctive operators.
- If A is β-formula B, by Condition (3) $B_1 \in U$ or $B_2 \in U$. By the inductive hypothesis, $\mathcal{I} \models B_1$ or $\mathcal{I} \models B_2$, so $\mathcal{I} \models A$ by definition of the disjunctive operators.

$\square$

Step-4: Note that the formula labeling the root is in the set.

Proof of Completeness:

- Let $\mathcal{T}$ be a completed open tableau for A.
- Then U, the union of the labels of the nodes on an open branch, is a Hintikka set by Theorem 37.
- Theorem 38 shows an interpretation $\mathcal{I}$ can be found such that U is simultaneously satisfiable in $\mathcal{I}$.
- A, the formula labeling the root, is an element of U so $\mathcal{I} \models A$.

$\square$

Summary

1. How one constructs an FOL formula. Variables, terms, function symbols, predicate symbols, atoms, literals, logical connectives, quantifiers
2. What an FOL formula means. Truth values true and false. Interpretations: domain and assignments
3. Whether an FOL formula evaluates to true under any or all interpretations. Semantic argument method
4. Substitution, which is a tool for manipulating formulae and making general claims. Safe and schema substitutions. Substitution of equivalent formulae. Valid schemata
5. A normal form is a set of syntactically restricted formulae such that every FOL formula is equivalent to some member of the set
6. A review of decidability, complexity theory and meta-theorems.