Predicate Logic

Bow-Yaw Wang

Institute of Information Science
Academia Sinica, Taiwan

October 7, 2017

Outline

1. The need for a richer language
2. Predicate logic as a formal language
3. Proof theory of predicate logic
4. Quantifier equivalences
5. Semantics of predicate logic
6. Undecidability of predicate logic
7. Expressiveness of predicate logic
8. The Coq Proof Assistant

Outline

Limitation of Propositional Logic

- Consider the following sentence:

  Every student is younger than some instructor.
  （每位學生都比某位授課者年輕。）
- How do we express it in propositional logic?
  - What are propositional atoms?
- To express the sentence, let us define some predicates.
  - Informally, a predicate is a function from objects to truth values.
- For example, $S(andy)$ denotes that Andy is a student; $I(paul)$ denotes that Paul is an instructor; $Y(andy, paul)$ denotes that Andy is younger than Paul.
- We also use variables to denote an object.
  - $S(x)$ means x is a student; $I(x)$ means x is an instructor; $Y(x, y)$ means x is younger than y.
- Here is a predicate logic formula expressing the sentence:

$$\forall x(S(x) \implies (\exists y(I(y) \wedge Y(x, y)))).$$

More Examples

- "Not all birds can fly."
  - Let $B(x)$ denote x is a bird, and $F(x)$ denote x can fly.
  - $\neg(\forall x(B(x) \implies F(x)))$.
- "Some bird cannot fly."

$$\exists x(B(x) \land \neg F(x)).$$

- Do "not all birds can fly" and "some bird cannot fly" have the same meaning?
  - What are the "meaning" of these sentences?
  - What is the "same"?

More Examples

- "Andy and Paul have the same biological maternal grandmother."
  - Let $M(x, y)$ denote that x is y's mother.
  - Consider
    $$\forall x \forall y \forall u \forall v (M(x, y) \wedge M(y, andy) \wedge M(u, v) \wedge M(v, paul) \implies x = u).$$
  - Let $m(x)$ denote x's biological mother.
  - Consider
    $$m(m(andy)) = m(m(paul)).$$
- Since everyone has exactly one biological mother, we introduce a function $m(x)$ to denote this fact.
- In this chaper, we will consider these questions formally.

Outline

Syntax

- In our examples, there are two sorts of things:
  - $B(x)$, $M(x,y)$, $B(x) \wedge \neg F(x)$ are formulae. They denote truth values;
  - y, $paul$, $m(x)$ are terms. They denote objects.
- Hence a predicate vocabulary has three sets.
- $\mathcal{P}$ is a set of predicate symbols ($B(x), M(x,y)$ etc).
- $\mathcal{F}$ is a set of function symbols ($m(x)$ etc).
- $\mathcal{C}$ is a set of constant symbols ($andy, paul$ etc).
- A function symbol $f \in \mathcal{F}$ with arity n (or n-arity) takes n arguments.
- Observe that a 0-arity (or nullary) function is in fact a constant.
- Hence $\mathcal{C} \subseteq \mathcal{F}$. We can ignore $\mathcal{C}$ for convenience.

Outline

Terms

Definition

Terms are defined as follows.

- Any variable is a term;
- If $c \in \mathcal{F}$ is a nullary function symbol, c is a term;
- If $t_1, t_2, \ldots, t_n$ are terms and $f \in \mathcal{F}$ has arity $n > 0$, then $f(t_1, t_2, \ldots, t_n)$ is a term;
- Nothing else is a term.

- In Backus Naur form, we have

$$t ::= x \mid c \mid f(t, \ldots, t)$$

where $x \in \mathrm{var}$ is a variable, $c \in \mathcal{F}$ a nullary function symbol, and $f \in \mathcal{F}$ a function symbol with arity > 0.

Terms

- Let $n, f, g \in \mathcal{F}$ be function symbols with arity 0, 1, and 2 respectively.
- $g(f(n), n)$, $f(f(n))$, $f(g(n, g(f(n), n)))$ are terms.
- $g(n)$, $f(n, n)$, $n(g)$ are not terms.
- Let $0, 1, \ldots$ be nullary function symbols, and $+, -, \times$ binary function symbols.
- $+(\times(3, x), 1)$, $+(\times(x, x), +(\times(2, \times(x, y))), \times(y, y))$ are terms.
- In infix notation, they are $(3 \times x) + 1$, $(x \times x) + ((2 \times (x \times y)) + (y \times y))$.

Outline

1. The need for a richer language
2. Predicate logic as a formal language
   - Terms
   - Formulae
   - Free and bound variables
   - Substitution
3. Proof theory of predicate logic
4. Quantifier equivalences
5. Semantics of predicate logic
6. Undecidability of predicate logic

Formulae

Definition

Formulae are defined as follows.

- If $P \in \mathcal{P}$ is a predicate symbol with arity $n \geq 1$, and $t_1, t_2, \ldots t_n$ are terms over $\mathcal{F}$, then $P(t_1, t_2, \ldots, t_n)$ is a formula;
- If ϕ is a formula, so is $(\neg\phi)$;
- If ϕ and ψ are formulae, so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, and $(\phi \implies \psi)$.
- If ϕ is a formula and x is a variable, then $(\forall x \phi)$ and $(\exists x \phi)$ are formulae;
- Nothing else is a formula.

- In Backus Naur form, we have

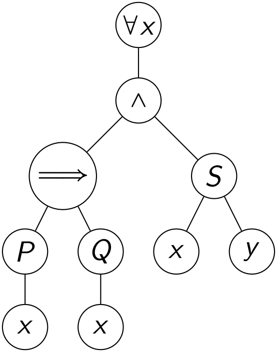
$$\phi ::= P(t_1, \ldots, t_n) \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \implies \phi) \mid (\forall x \phi) \mid (\exists x \phi)$$

where $P \in \mathcal{P}$ is a predicate symbol of arity n, $t_1, \ldots, t_n$ terms over $\mathcal{F}$, and $x \in \text{var}$ a variable.

Convention

- It is very tedious to write parentheses.
- We will assume the following binding priorities.
  - $\neg$, $\forall x$ and $\exists x$ (tightest)
  - $\vee$, $\wedge$
  - $\implies$ (right-associative and loosest)

Parse Tree



- A predicate logic formula can be represented as a parse tree.
  - $\forall x$, $\exists y$ are nodes;
  - arguments of function symbols are also nodes.
- The above figure gives the parse tree of $\forall x((P(x) \implies Q(x)) \wedge S(x, y))$.

Example

Example

Write "every son of my father is my brother" in predicate logic.

Proof.

Let me denote 'me', $S(x, y)$ (x is a son of y), $F(x, y)$ (x is the father of y), and $B(x, y)$ (x is a brother of y) be predicate symbols of arity 2. Consider

$$\forall x \forall y (F(x, me) \land S(y, x) \implies B(y, me)).$$

Alternatively, let f ($f(x)$ is the father of x) be a unary function symbol. Consider

$$\forall x (S(x, f(me)) \implies B(x, me)).$$

□

- Translating an English sentence into predicate logic can be tricky.
- Can you identify problem(s) in the example?

Outline

1. The need for a richer language
2. Predicate logic as a formal language
   - Terms
   - Formulae
   - Free and bound variables
   - Substitution
3. Proof theory of predicate logic
4. Quantifier equivalences
5. Semantics of predicate logic
6. Undecidability of predicate logic

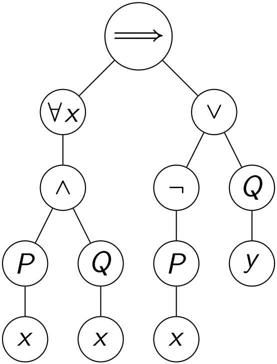
Constants and Variables

- Let c, d be constants (nullary functions).
- Consider $\forall x(P(x) \implies Q(x)) \wedge P(c) \implies Q(c)$.
  - If $P(x)$ implies $Q(x)$ for all x and $P(c)$ is true, then $Q(c)$ is true.
- Intuitively, $\forall y(P(y) \implies Q(y)) \wedge P(c) \implies Q(c)$ should have the same meaning.
- $\forall y(P(y) \implies Q(y)) \wedge P(d) \implies Q(d)$ is different.
  - We do not know if $Q(c)$ is true.
- Things can get very complicated when there are several variables.
  - $\forall x((P(x) \implies Q(x)) \wedge S(x, y))$
  - $\forall z((P(z) \implies Q(z)) \wedge S(z, y))$
  - $\forall y((P(y) \implies Q(y)) \wedge S(y, x))$

Free and Bound Variables

Definition

Let ϕ be a predicate logic formula. An occurrence of x in ϕ is free in ϕ if it is a leaf node without ancestor nodes $\forall x$ or $\exists x$ in the parse tree of ϕ. Otherwise, the occurrence of x is bound. The scope of $\forall x$ in $\forall x \phi$ is the formula ϕ minus any subformula in ϕ of the form $\forall x \psi$ or $\exists x \psi$.



$$(\forall x (P(x) \wedge Q(x))) \implies (\neg P(x) \vee Q(y))$$

Outline

Subsitution

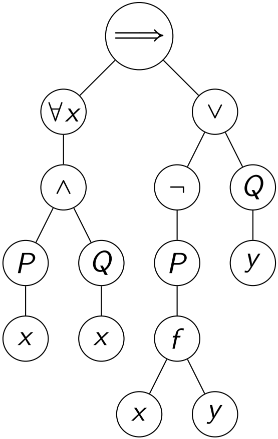
- Variables denote objects in predicate logic.
- Hence variables can be replaced by terms (but not formulae).
  - Replace x in $x \neq x + 1$ by 2 to get $2 \neq 2 + 1$.
  - What if we replace x by $2 = 2$?
- However, bound variables should not be replaced.
- The variables x and y in $\forall x \phi$ and $\exists y \psi$ denote <u>all</u> or <u>some</u> objects respectively.
  - What if we replace x in $\exists x(x = 0)$ by 1?

Definition

Given a variable x, a term t and a formula ϕ, define $\phi[t/x]$ to be the formula obtained by <u>replacing each free occurrence of x in ϕ with t</u>.

Example

- Let $\phi = (\forall x(P(x) \wedge Q(x))) \implies (\neg P(x) \vee Q(y))$. Consider $\phi[f(x,y)/x]$.



$$(\forall x(P(x) \wedge Q(x))) \implies (\neg P(x) \vee Q(y))[f(x,y)/x]$$

Variable Capture in Substitution

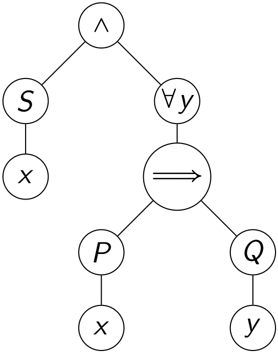
- Let $\phi = \exists y(y < x)$ and $\psi = \exists z(z < x)$.
  - Since ϕ and ψ only differ in bound variables, they should have the same meaning.
- Consider $\phi[(y - 1)/x] = \exists y(y < y - 1)$.
- The variable y in $y - 1$ is caught by the bound variable in ϕ.
- Consider $\psi[(y - 1)/x] = \exists z(z < y - 1)$.
- The variable y in $y - 1$ is not caught in the substitution $\psi[(y - 1)/x]$.

Definition

Let t be a term, x a variable, and ϕ a formula. t is free for x in ϕ if no free x leaf in ϕ occurs in the scope of $\forall y$ or $\exists y$ for any variable y occurring in t.

- Examples: $y - 1$ is free for x in $\exists z(z < x)$; $y - 1$ is not free for x in $\exists y(y < x)$.

Example



- Consider $\phi = S(x) \wedge \forall y(P(x) \implies Q(y))$ and $t = f(y, y)$.
- The two occurrences of x in ϕ are free.
- The right occurrence of x in ϕ is in the scope of $\forall y$ and y occurs in t.
- t is not free for x in ϕ.

Substitution and Variable Capture

- When t is not free for x in ϕ, the substitution $\phi[t/x]$ is not desirable.
- However, we can always rename bound variables for substitution.
- When we write $\phi[t/x]$, we mean all bound variables in ϕ are renamed so that t is free for x in ϕ.
- Examples.
  - $\phi = \exists y(y < x)$ and $t = y - 1$. t is not free for x in ϕ. Rename the bound variable y to z and obtain $\psi = \exists z(z < x)$. t is free for x in ψ.
  - $\phi = S(x) \land \forall y(P(x) \implies Q(y))$ and $t = f(y, y)$. t is not free for x in ϕ. Rename the bound variable y to z and obtain $\psi = S(x) \land \forall z(P(x) \implies Q(z))$. t is free for x in ψ.

Outline

Natural Deduction for Predicate Logic

- Similar to propositional logic, predicate logic has its natural deduction proof system.
- Naturally, the natural deduction proof rules for contradiction ($\bot$), negation ($\neg$), and Boolean connectives ($\vee$, $\wedge$, $\implies$) are the same as those in propositional logic.
- Additionally, there are proof rules for equality ($=$) and quantification ($\forall$ and $\exists$).
- Again, these additional rules have two types: introduction and elimination rules.

Equality

- Let s and t be terms.
- What do we mean by $s = t$?
- Shall we say $2 + 1 = 2 + 1$?
- What about $2^{61} - 1 = 2305843009213693951$?
- Apparently, if two terms are syntactically equal, they are equal.
  - This is called intensional equality.
- In practice, if two terms denote the same object, they are equal.
  - This is called extensional equality.

Natural Deduction Proof Rules for Equality

- The introduction rule for equality is as follows.

$$\frac{}{t = t} = i$$

- The elimination rule for equality is as follows.

$$\frac{t_1 = t_2 \quad \phi[t_1/x]}{\phi[t_2/x]} = e$$

  (t_1 and t_2 are free for x in ϕ).
  - The requirement "t_1 and t_2 are free for x in ϕ" is called the side condition of the proof rule.
- By convention, we assume the side condition holds in all substitutions.

Example

Example

Show

$x + 1 = 1 + x, (x + 1) > 1 \implies (x + 1) > 0 \vdash (1 + x) > 1 \implies (1 + x) > 0.$

Proof.

1	$x + 1 = 1 + x$	premise
2	$(x + 1) > 1 \implies (x + 1) > 0$	premise
3	$(1 + x) > 1 \implies (1 + x) > 0$	$=$e 1, 2

In step 3, take $\phi = \underline{x > 1 \implies x > 0}$, $t_1 = x + 1$, and $t_2 = 1 + x$. Then
$\phi[t_1/x] = (x + 1) > 1 \implies (x + 1) > 0$,
$\phi[t_2/x] = (1 + x) > 0 \implies (1 + x) > 0$. □

Reflexivity of Equality

Example

Show $t_1 = t_2 \vdash t_2 = t_1$.

Proof.

1	$t_1 = t_2$	premise
2	$t_1 = t_1$	$=$i
3	$t_2 = t_1$	$=$e, 1, 2

Take $\phi = (x = t_1)$. $\phi[t_1/x] = (t_1 = t_1)$ and $\phi[t_2/x] = (t_2 = t_1)$. $\square$

Transitivity of Equality

Example

Show $t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3$.

Proof.

1	$t_2 = t_3$	premise
2	$t_1 = t_2$	premise
3	$t_1 = t_3$	$=$e, 1, 2

Take $\phi = (t_1 = x)$. $\phi[t_2/x] = (t_1 = t_2)$ and $\phi[t_3/x] = (t_1 = t_3)$. $\square$

- Thus, the rules $=$i and $=$e give us the reflexity, symmetry, and transitivity of equality.

Natural Deduction Proof Rules for Universal Quantification

- The elimination rule for universal quantification is the following:

$$\frac{\forall x \phi}{\phi[t/x]} \; \forall x e$$

  when t is free for x in ϕ.

- To see why t must be free for x in ϕ, let ϕ be $\exists y(x < y)$. For natural numbers, $\forall x \exists y(x < y)$ is clearly true ("for any number, there is a larger number"). But if we take $t = y$, $\phi[t/x] = \exists y(y < y)$. This is wrong. Hence t must be free for x in ϕ.
  - If we really need to replace x by y in this case, we should rewrite $\exists y(x < y)$ to $\exists z(x < z)$ and obtain $\exists z(x < z)[x/y] = \exists z(y < z)$.

Natural Deduction Proof Rules for Universal Quantification

- The introduction rule for universal quantification opens a new box for a fresh variable x_0:

$$\dfrac{\boxed{\begin{array}{l} x_0 \\ \quad\vdots \\ \quad \phi[x_0/x] \end{array}}}{\forall x \phi} \; \forall x i$$

  (By "fresh," we mean x_0 does not occur outside of the box.)

- Informally, the rule $\forall x i$ says "if we can establish $\phi[x_0/x]$ for a fresh x_0, then we can derive $\forall x \phi$."
  - Intuitively, x_0 can be an arbitrary term since it is fresh and assumes nothing. If we can show $\phi[x_0/x]$, we have $\forall x \phi$.
  - Another way to see this is to replace x_0 by a term t in the box. We would have a proof for $\phi[t/x]$. That is, we have shown $\forall x \phi$.

Example

Example

Show $\forall x(P(x) \implies Q(x)), \forall x P(x) \vdash \forall x Q(x)$.

Proof.

1		$\forall x(P(x) \implies Q(x))$	premise
2		$\forall x P(x)$	premise
3	x_0	$P(x_0) \implies Q(x_0)$	$\forall x$e 1
4		$P(x_0)$	$\forall x$e 2
5		$Q(x_0)$	$\implies$e 4, 3
6		$\forall x Q(x)$	$\forall x$i 3–5

$\square$

Example

Example

Show $P(t), \forall x(P(x) \implies \neg Q(x)) \vdash \neg Q(t)$ for any term t.

Proof.

1	$P(t)$	premise
2	$\forall x(P(x) \implies \neg Q(x))$	premise
3	$P(t) \implies \neg Q(t)$	$\forall x$e 2
4	$\neg Q(t)$	$\implies$ e 1, 3

$\square$

- In step 3, we apply $\forall x$e by replacing x with t. We could apply the same rule with a different term, say, a. Hence the rule $\forall x$e is in fact a scheme of rules; one for each term t (free of x in ϕ).
- Also, we have different introduction and elimination rule. for different variables. That is, we have $\forall x$i, $\forall x$e, $\forall y$i, $\forall y$e, and so on. We will simply write $\forall$i and $\forall$ e when bound variables are clear.

Universal Quantification and Conjunction

- It is helpful to compare proof rules for universal quantification and conjunction.
- Introduction rules:
  - To establish $\forall x \phi$, we need to show $\phi[t/x]$ for any term t. This is accomplished by proving $\phi[x_0/x]$ with the box for a fresh variable x_0;
  - To establish $\phi \wedge \psi$, we need to show ϕ and ψ.
- Elimination rules:
  - To eliminate $\forall x \phi$, we pick a term (free for x in ϕ) and deduce $\phi[t/x]$;
  - To eliminate $\phi \wedge \psi$, we deduce ϕ (or ψ).

Natural Deduction Proof Rule for Existential Quantification

- The introduction rule for existential quantification is as follows.

$$\frac{\phi[t/x]}{\exists x \phi}\ \exists x i$$

  when t is free for x in ϕ.
- To see why t must be free for x in ϕ, consider $\exists x \forall y(x = y)$. This is clearly wrong for, say, natural numbers. Let $\phi = \forall y(x = y)$ and $t = y$. Since $\phi[t/x] = \forall y(y = y)$ is deducible (=i, $\forall y$i), we would have $\exists x \forall y(x = y)$.
- Recall the elimination rule for universal quantification:

$$\frac{\forall x \phi}{\phi[t/x]}\ \forall x e$$

  when t is free for x in ϕ.
- $\forall x$e is the "dual" of $\exists x$i.
  - Recall the duality of $\wedge$e and $\vee$i.

Natural Deduction Proof Rule for Existential Quantification

- The elimination rule for existential quantification is as follows.

$$\frac{\exists x \phi \qquad \boxed{\begin{array}{ll} x_0 & \\ & \phi[x_0/x] \\ & \quad\vdots \\ & \quad\chi \end{array}}}{\chi}\ \exists x e$$

- Informally, the rule $\exists x e$ says: to show χ from $\exists x \phi$, we show χ by assuming $\phi[x_0/x]$ for a fresh variable x_0.
  - Intuitively, x_0 stands for an unknown term t such that $\phi[t/x]$ holds. If we can deduce χ by assuming $\phi[t/x]$, then χ is deducible from $\exists x \phi$.
- Note that x_0 must not occur in χ.

Existential Quantification and Disjunction

- It is helpful to compare the elimination rules for existential quantification and disjunction.
- Recall

$$\frac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi \\ \vdots \\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi \\ \vdots \\ \chi\end{array}}}{\chi} \vee e$$

- To eliminate $\phi \vee \psi$, we show that χ is deducible by assuming ϕ or assuming ψ.
- To eliminate $\exists x \phi$, we show that χ is deducible by assuming $\phi[x_0/x]$.

Subformula Property I

- An elimination rule has subformula property if it must conclude with a subformula of the eliminated formula.
- For example, both $\wedge e_1$ and $\neg e$ have the subformula property.

$$\frac{\phi \wedge \psi}{\phi}\ \wedge e_1 \qquad \frac{\neg\neg\phi}{\phi}\ \neg\neg e$$

- Since the conclusion of $\forall x e$ has the same logical structure as the eliminated formula, we also say $\forall x e$ has the subformula property.

$$\frac{\forall x \phi}{\phi[t/x]}\ \forall x e$$

  - Strictly speaking, $\phi[t/x]$ may not be a subformula of $\forall x \phi$.

Subformula Property II

- The subformula property helps proof search.
  - We need not invent a formula for rules with the property.
  - Such rules are good for automated proof search.
- $\lor$e and $\exists x$e however do not have the subformula property.

$$\dfrac{\phi \lor \psi \qquad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \qquad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}\ \lor e \qquad\qquad \dfrac{\exists x \phi \qquad \boxed{\begin{array}{ll} x_0 & \\ & \phi[x_0/x] \\ & \vdots \\ & \chi \end{array}}}{\chi}\ \exists x e$$

  - The conclusion χ must be chosen carefully.

Examples I

Example

Show $\forall x \phi \vdash \exists x \phi$.

Proof.

1	$\forall x \phi$	premise
2	$\phi[x/x]$	$\forall x$e 1
3	$\exists x \phi$	$\exists x$i 2

(Is x free for x in $\phi[x/x]$?) □

- Is it correct?

Examples II

Example

Show $\forall x(P(x) \implies Q(x)), \exists x P(x) \vdash \exists x Q(x)$.

Proof.

1		$\forall x(P(x) \implies Q(x))$	premise
2		$\exists x P(x)$	premise
3	x_0	$P(x_0)$	assumption
4		$P(x_0) \implies Q(x_0)$	$\forall x$e 1
5		$Q(x_0)$	$\implies$e 3, 4
6		$\exists x Q(x)$	$\exists x$i 5
7		$\exists x Q(x)$	$\exists x$e 2, 3–6

(Can we close the box at line 5 instead of 6? Why not?) □

Examples III

Example

Show $\exists x P(x), \forall x \forall y (P(x) \implies Q(y)) \vdash \forall y Q(y)$.

Proof.

1		$\exists x P(x)$	premise
2		$\forall x \forall y (P(x) \implies Q(y))$	premise
3	y_0		
4	x_0	$P(x_0)$	assumption
5		$\forall y (P(x_0) \implies Q(y))$	$\forall x$e 2
6		$P(x_0) \implies Q(y_0)$	$\forall y$e 5
7		$Q(y_0)$	$\implies$e 4, 6
8		$Q(y_0)$	$\exists x$e 1, 4–7
9		$\forall y Q(y)$	$\forall y$i 3–8

□

Box Box Box I

- Fresh variables in box must not appear outside!
- If not, we could show $\exists x P(x), \forall x (P(x) \implies Q(x)) \vdash \forall y Q(y)$!

1		$\exists x P(x)$	premise
2		$\forall x (P(x) \implies Q(x))$	premise
3	x_0		
4	x_0	$P(x_0)$	assumption
5		$P(x_0) \implies Q(x_0)$	$\forall x$e 2
6		$Q(x_0)$	$\implies$ e 4, 5
7		$Q(x_0)$	$\exists x$e 1, 4–6
8		$\forall y Q(y)$	$\forall y$i 3–7

Outline

Equivalent Predicate Logic Formulae I

- Let ϕ and ψ be predicate logic formulae.
- $\phi \dashv\vdash \psi$ denotes tha $\phi \vdash \psi$ and $\psi \vdash \phi$.

Equivalent Predicate Logic Formulae II

Theorem

Let ϕ and ψ be predicate logic formulae. We have

1. (a) $\neg\forall x\phi \dashv\vdash \exists x\neg\phi$; (b) $\neg\exists x\phi \dashv\vdash \forall x\neg\phi$.
2. *When x is not free in ψ:*
   (a) $\forall x\phi \land \psi \dashv\vdash \forall x(\phi \land \psi)$; (b) $\forall x\phi \lor \psi \dashv\vdash \forall x(\phi \lor \psi)$;
   (c) $\exists x\phi \land \psi \dashv\vdash \exists x(\phi \land \psi)$; (d) $\exists x\phi \lor \psi \dashv\vdash \exists x(\phi \lor \psi)$;
   (e) $\forall x(\psi \implies \phi) \dashv\vdash \psi \implies \forall x\phi$;
   (f) $\exists x(\phi \implies \psi) \dashv\vdash \forall x\phi \implies \psi$;
   (g) $\forall x(\phi \implies \psi) \dashv\vdash \exists x\phi \implies \psi$;
   (h) $\exists x(\psi \implies \phi) \dashv\vdash \psi \implies \exists x\phi$.
3. (a) $\forall x\phi \land \forall x\psi \dashv\vdash \forall x(\phi \land \psi)$; (b) $\exists x\phi \lor \exists x\psi \dashv\vdash \exists(\phi \lor \psi)$.
4. (a) $\forall x\forall y\phi \dashv\vdash \forall y\forall x\phi$ (b) $\exists x\exists y\phi \dashv\vdash \exists y\exists x\phi$.

$\neg\forall x\phi \vdash \exists x\neg\phi$

1		$\neg\forall x\phi$	premise
2		$\neg\exists x\neg\phi$	assumption
3	x_0		
4		$\neg\phi[x_0/x]$	assumption
5		$\exists x\neg\phi$	$\exists x$i 4
6		$\bot$	$\neg$e 5, 2
7		$\phi[x_0/x]$	PBC 4–6
8		$\forall x\phi$	$\forall x$i 3–7
9		$\bot$	$\neg$e 8, 1
10		$\exists x\neg\phi$	PBC 2–9

- The proof structure is similar to $\neg(p_1 \wedge p_2) \vdash \neg p_1 \vee \neg p_2$.

$\neg(p_1 \land p_2) \vdash \neg p_1 \lor \neg p_2$

1	$\neg(p_1 \land p_2)$	premise
2	$\neg(\neg p_1 \lor \neg p_2)$	assumption
3	$\neg p_1$	assumption
4	$\neg p_1 \lor \neg p_2$	$\lor i_1$ 3
5	$\bot$	$\neg e$ 4, 2
6	p_1	PBC 3–5
3'	$\neg p_2$	assumption
4'	$\neg p_1 \lor \neg p_2$	$\lor i_2$ 3'
5'	$\bot$	$\neg e$ 4', 2'
6'	p_2	PBC 3'–5'
7	$p_1 \land p_2$	$\land i$ 6, 6'
8	$\bot$	$\neg e$ 7, 1
9	$\neg p_1 \lor \neg p_2$	PBC 2–8.

$\exists x \neg\phi \vdash \neg\forall x \phi$

1		$\exists x \neg\phi$	premise
2		$\forall x \phi$	assumption
3	x_0	$\neg\phi[x_0/x]$	assumption
4		$\phi[x_0/x]$	$\forall$e 2
5		$\bot$	$\neg$e 4, 3
6		$\bot$	$\exists x$e 1, 3–5
7		$\neg\forall x \phi$	$\neg$i 2–6

$\forall x\phi \wedge \psi \vdash \forall x(\phi \wedge \psi)$ and $\forall x(\phi \wedge \psi) \vdash \forall x\phi \wedge \psi$ (x not free in ψ)

1		$(\forall x\phi) \wedge \psi$	premise
2		$\forall x\phi$	$\wedge e_1$ 1
3		ψ	$\wedge e_2$ 2
4	x_0		
5		$\phi[x_0/x]$	$\forall x$e 2
6		$\phi[x_0/x] \wedge \psi$	$\wedge$i 5, 3
7		$(\phi \wedge \psi)[x_0/x]$	x not free in ψ
8		$\forall x(\phi \wedge \psi)$	$\forall x$i 4–7
1		$\forall x(\phi \wedge \psi)$	premise
2	x_0		
3		$(\phi \wedge \psi)[x_0/x]$	$\forall x$e 1
4		$\phi[x_0/x] \wedge \psi$	x not free in ψ
5		ψ	$\wedge e_2$ 4
6		$\phi[x_0/x]$	$\wedge e_1$ 4
7		$\forall x\phi$	$\forall x$i 2–6
8		$\forall x\phi \wedge \psi$	$\wedge$i 7, 5

$(\exists x\phi) \vee (\exists x\psi) \vdash \exists x(\phi \vee \psi)$

1		$(\exists x\phi) \vee (\exists x\psi)$	premise
2		$\exists x\phi$	assumption
3	x_0	$\phi[x_0/x]$	assumption
4		$\phi[x_0/x] \vee \psi[x_0/x]$	$\vee\mathrm{i}_1$ 3
5		$(\phi \vee \psi)[x_0/x]$	same as 4
6		$\exists x(\phi \vee \psi)$	$\exists x\mathrm{i}$ 5
7		$\exists x(\phi \vee \psi)$	$\exists x\mathrm{e}$ 2, 3–6
2'		$\exists x\psi$	assumption
3'	y_0	$\psi[y_0/x]$	assumption
4'		$\phi[y_0/x] \vee \psi[y_0/x]$	$\vee\mathrm{i}_2$ 3'
5'		$(\phi \vee \psi)[y_0/x]$	same as 4'
6'		$\exists x(\phi \vee \psi)$	$\exists x\mathrm{i}$ 5'
7'		$\exists x(\phi \vee \psi)$	$\exists x\mathrm{e}$ 2', 3'–6'
8		$\exists x(\phi \vee \psi)$	$\vee\mathrm{e}$ 1, 2–7, 2'–7'

$\exists x \exists y \phi \vdash \exists y \exists x \phi$

1		$\exists x \exists y \phi$	premise	
2	x_0	$(\exists y \phi)[x_0/x]$	assumption	
3		$\exists y (\phi[x_0/x])$	x and y different	
4	y_0	$\phi[x_0/x][y_0/y]$	assumption	
5		$\phi[y_0/y][x_0/x]$	x, y, x_0, y_0 different	
6		$\exists x \phi[y_0/y]$	$\exists x$i 5	
7		$\exists y \exists x \phi$	$\exists y$i 6	
8		$\exists y \exists x \phi$	$\exists y$e 3, 4–7	
9		$\exists y \exists x \phi$	$\exists x$e 1, 2–8	

Outline

Deduction and Satisfaction

- Let Γ be a set of predicate logic formulae and ψ a predicate logic formula.
- We know how to show $\Gamma \vdash \psi$.
  - Intuitively, ψ "holds" when every formulae in Γ hold.
- What if we want to show $\Gamma \nvdash \psi$?
  - How do we show "there is no such deduction?"
- Intuitively, we want to argue that ψ does not hold even when every formulae in Γ hold.
- Hence we will discuss when predicate logic formulae "hold."

Outline

Models

- Recall that we have constant, function, and predicate symbols in predicate logic.
- The semantics of terms and atomic predicates are defined in models.

Definition

Let $\mathcal{F}$ and $\mathcal{P}$ be a set of function and predicate symbols respectively. A model $\mathcal{M}$ of $(\mathcal{F}, \mathcal{P})$ consists of

- A non-empty set A called the universe;
- For function symbol $f \in \mathcal{F}$ with arity $n \geq 0$, a function $f^{\mathcal{M}} : A^n \to A$;
  - Particularly, a constant symbol $c \in \mathcal{F}$ is an element $c^{\mathcal{M}} \in A$.
- For predicate symbol $P \in \mathcal{P}$ with arity $n > 0$, a set $P^{\mathcal{M}} \subseteq A^n$.

Example of Models

- Let $\mathcal{F} = \{e, \cdot\}$ and $\mathcal{P} = \{\leq\}$ where e is a constant, $\cdot$ a binary function, and $\leq$ a binary predicate symbol respectively. We use infix notation for $\cdot$ and $\leq$.
- Consider the model $\mathcal{M}$:
  - the universe A is the set of all binary finite strings;
  - $e^{\mathcal{M}}$ is the empty string ϵ;
  - $\cdot^{\mathcal{M}}$ is string concatenation;
  - $\leq^{\mathcal{M}}$ is the string prefix relation.
- For instance, $00 \ \cdot^{\mathcal{M}} \ 111 = 00111$ and $01 \leq^{\mathcal{M}} 011$.
- In this model,
  - $\forall x((x \leq x \cdot e) \land (x \cdot e \leq x))$ is true.
  - $\exists y \forall x (y \leq x)$ is true.
  - $\forall x \forall y \forall z ((x \leq y) \implies (x \cdot z \leq y \cdot z))$ is false.

Environment

- For the semantics of $\forall x \phi$ and $\exists x \phi$, we need to check whether ϕ is true when x is assigned to an element of the universe.
- A model $(\mathcal{F}, \mathcal{P})$ however does not give semantics to variables.

Definition

An environment for a universe A is a function $l : \text{var} \to A$. If l is an environment, $x \in \text{var}$, and $a \in A$, the environment $l[x \mapsto a]$ is defined as follows.

$$l[x \mapsto a](y) = \begin{cases} a & \text{if } x = y \\ l(y) & \text{if } x \neq y \end{cases}$$

Semantics of Predicate Logic Formulae

Definition

Let $\mathcal{M}$ be a model of $(\mathcal{F}, \mathcal{P})$, l an environment, and ϕ a predicate logic formula. $\mathcal{M} \vDash_l \phi$ holds is defined as follows.

- $\mathcal{M} \vDash_l P(t_1, t_2, \ldots, t_n)$ holds if $(a_1, a_2, \ldots, a_n) \in P^{\mathcal{M}}$ where $a_1, a_2, \ldots, a_n \in A$ are computed for $t_1, t_2, \ldots, t_n$ by $\mathcal{F}$ and l;
- $\mathcal{M} \vDash_l \forall x \psi$ holds if $\mathcal{M} \vDash_{l[x \mapsto a]} \psi$ for every $a \in A$;
- $\mathcal{M} \vDash_l \exists x \psi$ holds if $\mathcal{M} \vDash_{l[x \mapsto a]} \psi$ for some $a \in A$;
- $\mathcal{M} \vDash_l \neg \psi$ holds if it is not the case $\mathcal{M} \vDash_l \psi$;
- $\mathcal{M} \vDash_l \psi_0 \vee \psi_1$ holds if $\mathcal{M} \vDash_l \psi_0$ holds or $\mathcal{M} \vDash_l \psi_1$ holds;
- $\mathcal{M} \vDash_l \psi_0 \wedge \psi_1$ holds if $\mathcal{M} \vDash_l \psi_0$ holds and $\mathcal{M} \vDash_l \psi_1$ holds;
- $\mathcal{M} \vDash_l \psi_0 \implies \psi_1$ holds if $\mathcal{M} \vDash_l \psi_1$ holds whenever $\mathcal{M} \vDash_l \psi_0$ holds.

If $\mathcal{M} \vDash_l \phi$ holds, we say ϕ computes to T in $\mathcal{M}$ with respect to l. Also, we write $\mathcal{M} \nvDash_l \phi$ when it is not the case $\mathcal{M} \vDash_l \phi$.

Sentences

- Let ϕ be a predicate logic formula, l and l' two environments that agree on free variables of ϕ.
  - That is, $l(x) = l'(x)$ for every free variable x in ϕ.
- By induction on ϕ, it is straightforward to show $\mathcal{M} \vDash_l \phi$ holds if and only if $\mathcal{M} \vDash_{l'} \phi$.
- A <u>sentence</u> is a predicate logic formula without free variables.
- Let ϕ be a sentence. Either
  - $\mathcal{M} \vDash_l \phi$ holds for every environment l; or
  - $\mathcal{M} \vDash_l \phi$ does not hold for every environment l.
- Hence we write $\mathcal{M} \vDash \phi$ (or $\mathcal{M} \nvDash \phi$) for a sentence ϕ since the choice of l does not matter.

Example

- Consider $(\mathcal{F}, \mathcal{P}) = (\{\text{alma}\}, \{\text{loves}\})$ where alma is a constant and loves is a binary predicate.
- Let $\mathcal{M}$ be a model of $(\mathcal{F}, \mathcal{P})$ with the universe $A = \{a, b, c\}$, $\text{alma}^{\mathcal{M}} = a$, and $\text{loves}^{\mathcal{M}} = \{(a, a), (b, a), (c, a)\}$.
- Consider the statement:

  None of Alma's lovers' lovers love her.

- We first translate the statement into a predicate logic formula ϕ:

$$\forall x \forall y (\text{loves}(x, \text{alma}) \wedge \text{loves}(y, x) \implies \neg\text{loves}(y, \text{alma})).$$

- We have $\mathcal{M} \nvDash \phi$.
  - Choose a for x and b for y. We have $(a, a) \in \text{loves}^{\mathcal{M}}$ and $(b, a) \in \text{loves}^{\mathcal{M}}$ but it is not the case $(b, a) \notin \text{loves}^{\mathcal{M}}$.

Outline

Semantic Entailment

Definition

Let Γ be a (possibly infinite) set of predicate logic formulae and ψ a predicate logic formula.

- $\Gamma \vDash \psi$ holds (or Γ semantically entails ψ) if for every model $\mathcal{M}$ and environment l, $\mathcal{M} \vDash_l \psi$ holds whenever $\mathcal{M} \vDash_l \phi$ holds for every $\phi \in \Gamma$;
- ψ is satisfiable if there is a model $\mathcal{M}$ and an environment l such that $\mathcal{M} \vDash_l \psi$ holds;
- ψ is valid if $\mathcal{M} \vDash_l \psi$ holds for every model $\mathcal{M}$ and environment l where we can compute ψ;
- Γ is consistent or satisfiable if there is a model $\mathcal{M}$ and an environment l such that $\mathcal{M} \vDash_l \phi$ for every $\phi \in \Gamma$.

- Note that "$\vDash$" has two different meanings:
  - $\mathcal{M} \vDash \psi$ means "ψ computes to T in $\mathcal{M}$;"
  - $\phi_1, \phi_2, \ldots, \phi_n \vDash \psi$ means "ψ is semantically entailed by $\phi_1, \phi_2, \ldots, \phi_n$."

Checking $\mathcal{M} \vDash \psi$ and $\phi_1, \phi_2, \ldots, \phi_n \vDash \psi$

- Let $\psi, \phi_1, \phi_2, \ldots, \phi_n$ be sentences.
- To check if $\mathcal{M} \vDash \psi$ holds, we need to enumerate all elements in the universe if ψ contains $\forall$ or $\exists$.
- To check if $\phi_1, \phi_2, \ldots, \phi_n \vDash \psi$ holds, we need to consider all possible models satisfying $\phi_1, \phi_2, \ldots, \phi_n$.
- Both sound difficult since a model may contain an infinite number of elements in its universe.
- However, we may still prove semantic entailments.

Examples I

Example

Show $\forall x(P(x) \implies Q(x)) \vDash \forall x P(x) \implies \forall x Q(x)$.

Proof.

Let $\mathcal{M}$ be a model that $\mathcal{M} \vDash \forall x(P(x) \implies Q(x))$. There are two cases:

- $\mathcal{M} \nvDash \forall x P(x)$. Then $\mathcal{M} \vDash \forall x P(x) \implies \forall x Q(x)$.
- $\mathcal{M} \vDash \forall x P(x)$. Let a be an element in the universe of $\mathcal{M}$. We have $a \in P^{\mathcal{M}}$ since $\mathcal{M} \vDash \forall x P(x)$ and hence $a \in Q^{\mathcal{M}}$ since $\mathcal{M} \vDash \forall x(P(x) \implies Q(x))$. That is, $\mathcal{M} \vDash \forall x Q(x)$. We conclude $\mathcal{M} \vDash \forall x P(x) \implies \forall x Q(x)$.

$\square$

Examples II

Example

Show $\forall x P(x) \implies \forall x Q(x) \not\models \forall x (P(x) \implies Q(x))$.

Proof.

Let $\mathcal{M}'$ be a model where $A' = \{a, b\}$, $P^{\mathcal{M}'} = \{a\}$, and $Q^{\mathcal{M}'} = \{b\}$. Since $\mathcal{M}' \not\models \forall x P(x)$, $\mathcal{M}' \models \forall x P(x) \implies \forall x Q(x)$. Since $a \in P^{\mathcal{M}'}$ but $a \notin Q^{\mathcal{M}'}$, $\mathcal{M}' \not\models \forall x (P(x) \implies Q(x))$. $\square$

Outline

Semantics of Equality

- Observe that = is also a binary predicate.
- But the symbol "=" is somewhat special.
  - We did not say $= \in \mathcal{P}$.
  - Rather, we explicitly say that = denotes the equality.
- This is because we do not want to interpret the equality arbitrarily.
  - It sounds absurd if $a = b$ means a is not b.
- In all model $\mathcal{M}$, we always have $=^{\mathcal{M}} \triangleq \{(a, a) : a \in A\}$.

Outline

Validity Problem for Predicate Logic

Definition

Given a predicate logic formula ϕ, the validity problem for predicate logic is to check whether $\vDash \phi$ holds or not.

- For a propositional logic formula ϕ, it is decidable to check whether $\vDash \phi$ holds.
  - The validity problem for propositional logic is coNP-complete.
- For a predicate logic formula ϕ, it is unclear how to design an algorithm.
- We will show the validity problem for predicate logic is undecidable.

Post Correspondence Problem

Definition

Given $C = ((s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k))$ where s_i, t_i are non-empty binary strings for every $1 \leq i \leq k$. The Post correspondence problem (PCP) is to check whether there are $1 \leq i_1, i_2, \ldots, i_n \leq k$ such that $s_{i_1} s_{i_2} \cdots s_{i_n} = t_{i_1} t_{i_2} \cdots t_{i_n}$.

- For example, consider $C = ((1, 101), (10, 00), (011, 11))$. We have

$$\underline{1}\,\underline{011}\,\underline{10}\,\underline{011} = \underline{101}\,\underline{11}\,\underline{00}\,\underline{11}.$$

- The Post correspondence problem is undecidable.
  - For details, study computational complexity.

Undecidability of Validity Problem I

Theorem

The validity problem for predicate logic is undecidable.

Proof.

Let $C = ((s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k))$ be an instance of PCP. We build a predicate logic formula ϕ so that C has a solution iff $\vDash \phi$ holds.
Let $\mathcal{F} = \{e, f_0, f_1\}$ and $\mathcal{P} = \{P\}$. The function symbols $e, f_0(), f_1()$ encode binary strings. The binary predicate symbol $P(s, t)$ means "there are $i_1, i_2, \ldots, i_m$ so that $s = s_{i_1} s_{i_2} \cdots s_{i_m}$ and $t = t_{i_1} t_{i_2} \cdots t_{i_m}$."
For instance, $1011 = f_1(f_1(f_0(f_1(e)))) = f_{1011}(e)$. Moreover, we write $f_{b_1 b_2 \cdots b_h}(v)$ for $f_{b_h}(f_{b_{h-1}} \cdots f_{b_1}(v))$ where $b_1 b_2 \cdots b_h$ is a binary string.

Proof (cont'd).

Define

$$\begin{aligned}
\phi_1 &\triangleq \bigwedge_{i=1}^{k} P(f_{s_i}(e), f_{t_i}(e)) \\
\phi_2 &\triangleq \forall v \forall w (P(v, w) \implies \bigwedge_{i=1}^{k} P(f_{s_i}(v), f_{t_i}(w)) \\
\phi_3 &\triangleq \exists z P(z, z)
\end{aligned}$$

We claim $\vDash \phi_1 \wedge \phi_2 \implies \phi_3$ iff C has a solution.

Suppose $\vDash \phi_1 \wedge \phi_2 \implies \phi_3$. Consider the model $\mathcal{M}$ for $(\mathcal{F}, \mathcal{P})$ as follows. The universe A is the set of all finite binary strings. $e^{\mathcal{M}} \triangleq \epsilon$, $f_0^{\mathcal{M}}(s) \triangleq s0$, and $f_1^{\mathcal{M}}(s) \triangleq s1$. Finally, $P^{\mathcal{M}} = \{(s, t) :$ there are $i_1, i_2, \ldots, i_m$ so that $s = s_{i_1} s_{i_2} \cdots s_{i_m}$ and $t = t_{i_1} t_{i_2} \cdots t_{i_m}\}$. We have $\mathcal{M} \vDash \phi_1 \wedge \phi_2 \implies \phi_3$. Moreover, since $\mathcal{M} \vDash \phi_1$ and $\mathcal{M} \vDash \phi_2$ (why?), $\mathcal{M} \vDash \phi_3$. That is, there is a binary string z and $i_1, i_2, \ldots, i_n$ such that $z = s_{i_1} s_{i_2} \cdots s_{i_n} = t_{i_1} t_{i_2} \cdots t_{i_n}$.

Proof (cont'd).

Conversely, suppose C has a solution $i_1, i_2, \ldots, i_n$ that $s_{i_1} s_{i_2} \cdots s_{i_n} = t_{i_1} t_{i_2} \cdots t_{i_n}$. We need to show $\mathcal{M}' \vDash \phi_1 \wedge \phi_2 \implies \phi_3$ for every model $\mathcal{M}'$ defining $e^{\mathcal{M}'}$, $f_0^{\mathcal{M}'}$, $f_1^{\mathcal{M}'}$, and $P^{\mathcal{M}'}$. Clearly, $\mathcal{M}' \vDash \phi_1 \wedge \phi_2 \implies \phi_3$ when $\mathcal{M}' \nvDash \phi_1 \wedge \phi_2$. It suffices to consider $\mathcal{M}' \vDash \phi_1 \wedge \phi_2$, and show $\mathcal{M}' \vDash \phi_3$ as well.

Let A' be the universe of $\mathcal{M}'$. We interpret finite binary strings in A' as follows.

$$\begin{array}{rcl} \text{interpret}(\epsilon) & \triangleq & e^{\mathcal{M}'} \\ \text{interpret}(s0) & \triangleq & f_0^{\mathcal{M}'}(\text{interpret}(s)) \\ \text{interpret}(s1) & \triangleq & f_1^{\mathcal{M}'}(\text{interpret}(s)). \end{array}$$

Hence, for instance, the string 1011 is interpreted as the element $f_1^{\mathcal{M}'}(f_1^{\mathcal{M}'}(f_0^{\mathcal{M}'}(f_1^{\mathcal{M}'}(e^{\mathcal{M}'}))))$. Generally, a finite binary string s is interpreted as $f_s^{\mathcal{M}'}(e^{\mathcal{M}'})$ in A'.

Undecidability of Validity Problem IV

Proof (cont'd).

Since $\mathcal{M}' \vDash \phi_1$, we have

$$(\text{interpret}(s_i), \text{interpret}(t_i)) \in P^{\mathcal{M}'} \text{ for } 1 \leq i \leq k.$$

Since $\mathcal{M}' \vDash \phi_2$, we have for every $(\text{interpret}(s), \text{interpret}(t)) \in P^{\mathcal{M}'}$,

$$(\text{interpret}(ss_i), \text{interpret}(tt_i)) \in P^{\mathcal{M}'} \text{ for } 1 \leq i \leq k.$$

Thus,

$$(\text{interpret}(s_{i_1} s_{i_2} \cdots s_{i_n}), \text{interpret}(t_{i_1} t_{i_2} \cdots t_{i_n})) \in P^{\mathcal{M}'}.$$

Moreover, $s_{i_1} s_{i_2} \cdots s_{i_n} = t_{i_1} t_{i_2} \cdots t_{i_n}$ since $i_1, i_2, \ldots, i_n$ is a solution to C. Hence $\text{interpret}(s_{i_1} s_{i_2} \cdots s_{i_n}) = \text{interpret}(t_{i_1} t_{i_2} \cdots t_{i_n})$. In other words, $\mathcal{M}' \vDash \phi_3$. □

Undecidability of Validity Problem V

Corollary

The satisfiability problem for predicate logic is undecidable.

Proof.

Observe $\vDash \phi$ holds iff $\neg\phi$ is not satisfiable. $\square$

Theorem

For any predicate logic sentence ϕ, $\vdash \phi$ iff $\vDash \phi$.

Corollary

It is undecideable to check whether $\vdash \phi$ for any predicate logic sentence ϕ.

- The undecidability of provability problem for predicate logic means it is impossible to build a perfect automatic theorem prover.
- Just like art, human creativity is still important in mathematics!

A Glimpse into Completeness I

- Similar to propositional logic, the natural deduction proof system for prediate logic is both sound and complete.
- Proving completeness however is much harder for predicate logic.
  - There is no truth table for predicate logic.
- We will give the first step to establish completeness.

A Glimpse into Completeness II

Lemma

Let Γ be a set of predicate logic formulae. The following are equivalent:

1. $\Gamma \vDash \phi$ *implies* $\Gamma \vdash \phi$;
2. $\Gamma \vDash \bot$ *implies* $\Gamma \vdash \bot$.

Proof.

(1) to (2). Suppose $\Gamma \vDash \bot$. Then $\Gamma \vdash \bot$ by (1).
(2) to (1). Suppose $\Gamma \vDash \phi$. Then $\Gamma \cup \{\neg\phi\} \vDash \bot$. Hence $\Gamma \cup \{\neg\phi\} \vdash \bot$. Therefore $\Gamma \vdash \phi$ using PBC. □

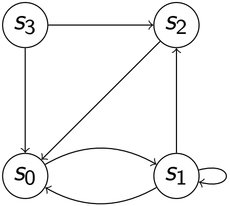
- To show completeness, it suffices to show that for every Γ, $\Gamma \nvdash \bot$ implies Γ is satisfiable.

Completeness and Undecidability

- We have two facts about predicate logic formulae.
  - $\models \phi$ implies $\vdash \phi$; and
  - it is undecidable to check if $\vdash \phi$.
- If a predicate logic formula is valid, then there is a natural deduction proof.
- On the other hand, it is impossible to have a program which checks whether there is a natural deduction proof.

Outline

Reachability



Example

Let $A = \{s_0, s_1, s_2, s_3\}$ and
$R^{\mathcal{M}} = \{(s_0, s_1), (s_1, s_0), (s_1, s_1), (s_1, s_2), (s_2, s_0), (s_3, s_0), (s_3, s_2)\}$. We write $s \to s'$ if $(s, s') \in R^{\mathcal{M}}$, and say there is a transition from s to s'.

Definition

Given a directed graph G and nodes n, n' in G, the reachability problem for G is to check whether there is a path of transition from n to n'.

Reachability in Predicate Logic

- Let $(\mathcal{F}, \mathcal{P}) = (\varnothing, \{R\})$ with a binary predicate R.
- A model of $(\mathcal{F}, \mathcal{P})$ denotes a directed graph.
- Can we write a predicate logic formula ϕ with free variables u and v to express $u \to \cdots \to v$?
- Consider

$$
\begin{array}{l}
u = v \vee \\
R(u, v) \vee \\
\exists x_0 (R(u, x_0) \wedge R(x_0, v)) \vee \\
\exists x_0 \exists x_1 (R(u, x_0) \wedge R(x_0, x_1) \wedge R(x_1, v)) \vee \cdots
\end{array}
$$

- But this is not a predicate logic formula since it is infinite.
- We will show it is impossible to express reachability in predicate logic.

Compactness Theorem

Theorem

Let Γ be a set of predicate logic sentences. If all finite subset of Γ is satisfiable, Γ is satisfiable.

Proof.

Assume Γ is not satisfiable. Then $\Gamma \vDash \bot$. By the completeness theorem for predicate logic, $\Gamma \vdash \bot$. Since deductions are finite, we have $\Delta \vdash \bot$ for some finite subset Δ of Γ. By the soundness theorem for predicate logic, $\Delta \vDash \bot$. Δ is not satisfiable, a contraction. $\square$

Löwenhein-Skolem Theorem

Theorem

Let ψ be a predicate logic sentence. If ψ has a model with at least n elements for every $n \geq 1$, ψ has a model with infinitely many elements.

Proof.

Define $\phi_n \triangleq \exists x_1 \exists x_2 \cdots \exists x_n \bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j)$. Let $\Gamma = \{\psi\} \cup \{\phi_n : n > 1\}$. For every finite subset Δ of Γ, Δ is satisfiable. By the compactness theorem, Γ is satisfiable by some model $\mathcal{M}$. Particularly, $\mathcal{M} \vDash \psi$ holds. Since $\mathcal{M} \vDash \phi_n$ for every $n \geq 1$, $\mathcal{M}$ has infinitely many elements. $\square$

Reachability in Predicate Logic

Theorem

There is no predicate logic formula ϕ with exactly two free variables u, v and exactly one binary predicate R such that ϕ holds in directed graphs iff there is a path in the graph from the node associated with u to the node associated with v.

Proof.

Suppose ϕ is a predicate logic formula expressing a path from u to v. Let c and c' be constants. Define $\phi_0 \triangleq c = c'$ and

$$\phi_n \triangleq \exists x_1 \exists x_2 \cdots \exists x_{n-1} (R(c, x_1) \wedge R(x_1, x_2) \wedge \cdots \wedge R(x_{n-1}, c')).$$

Then ϕ_n expresses that there is a path of length n from c to c'. Let $\Gamma = \{\phi[c/u][c'/v]\} \cup \{\neg\phi_i : i \geq 0\}$. For every finite subset Δ of Γ, Δ is satisfiable since there is always a path of an arbitrary finite length from c to c'. By the compactness theorem, Γ is satisfiable. A contradiction. □

Outline

Existential Second-Order Logic

- In predicate logic, we can ask if there is an element with a certain property.
  - Predicate logic is also called first-order logic.
- We can generalize the concept and ask if there is a predicate with a certain property in existential second-order logic.
- Let P be an n-ary predicate symbol.
- $\exists P \phi$ is an existential second-order logic formula.
- Let $\mathcal{M}$ be a model for all function and predicate symbols except P and $\mathcal{M}_T$ the same model with an additional n-ary relation $T (= P^{\mathcal{M}_T}) \subseteq A^n$. Define

$$\mathcal{M} \vDash_l \exists P \phi \text{ if } \mathcal{M}_T \vDash_l \phi \text{ for some } T (= P^{\mathcal{M}_T}) \subseteq A^n.$$

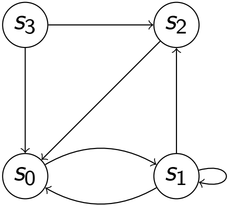
Unreachability in Existential Second-Order Logic I

- Consider the existential second-order logic formula $\exists P \forall x \forall y \forall z (C_1 \land C_2 \land C_3 \land C_4)$ where

$$
\begin{array}{lcl}
C_1 & \triangleq & P(x, x) \\
C_2 & \triangleq & P(x, y) \land P(y, z) \implies P(x, z) \\
C_3 & \triangleq & P(u, v) \implies \bot \\
C_4 & \triangleq & R(x, y) \implies P(x, y).
\end{array}
$$

  - C_i's are Horn clauses.

Unreachability in Existential Second-Order Logic II



- Consider the directed graph $\mathcal{M}$ in the previous slide.
- Let $l(u) = s_0$ and $l(v) = s_3$.
- Does $\mathcal{M} \vDash_l \exists P \forall x \forall y \forall z (C_1 \wedge C_2 \wedge C_3 \wedge C_4)$ hold?
  - Take $T \triangleq \{(s, s') \in A \times A : s' \neq s_3\} \cup \{(s_3, s_3)\}$.

Outline

Universal Second-Order Logic

- Let P be an n-ary predicate symbol.
- $\forall P \phi$ is a universal second-order logic formula.
- Let $\mathcal{M}$ be a model for all function and predicate symbols except P. Define

$$\mathcal{M} \vDash_l \forall P \phi \text{ if } \mathcal{M}_T \vDash_l \phi \text{ for every } T (= P^{\mathcal{M}_T}) \subseteq A^n.$$

Reachability in Universal Second-Order Logic I

Theorem

Let $\mathcal{M}$ be a model of $(\varnothing, \{R\})$ with a binary predicate symbol R. $\mathcal{M} \vDash_l \forall P \exists x \exists y \exists z(\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ holds iff $l(v)$ is R-reachable from $l(u)$ in $\mathcal{M}$, where $C_1 \triangleq P(x,x)$, $C_2 \triangleq P(x,y) \wedge P(y,z) \implies P(x,z)$, $C_3 \triangleq P(u,v) \implies \bot$, and $C_4 \triangleq R(x,y) \implies P(x,y)$.

Proof.

Assume $\mathcal{M}_T \vDash_l \exists x \exists y \exists z(\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ for every $T \subseteq A \times A$. Consider the reflexive and transitive closure T^* of $R^{\mathcal{M}}$. Then $\mathcal{M}_{T^*} \vDash_{l'} C_1 \wedge C_2 \wedge C_4$ where $l' = l[x,y,z \mapsto a,b,c]$ for some $a,b,c \in A^{\mathcal{M}_T}$. Hence $\mathcal{M}_{T^*} \vDash_{l'} \neg C_3$ and so $\mathcal{M}_{T^*} \vDash_{l'} P(u,v)$. In other words, $(l'(u), l'(v)) = (l(u), l(v)) \in T^*$. There is a finite path from $l(u)$ to $l(v)$.

Reachability in Universal Second-Order Logic II

Proof (cont'd).

Conversely, assume there is a finite path from $I(u)$ to $I(v)$. Let $T \subseteq A \times A$. There are two cases.

- T is not reflexive, not transitive, or does not contain $R^{\mathcal{M}}$. Then $\mathcal{M}_T \vDash_{I'} \neg C_1$, $\mathcal{M}_T \vDash_{I'} \neg C_2$, or $\mathcal{M}_T \vDash_{I'} \neg C_4$ for some $I' = I[x, y, z \mapsto a, b, c]$ for some $a, b, c \in A^{\mathcal{M}_T}$.
- T is reflexive, transitive, and contains $R^{\mathcal{M}}$. Then T contains the reflexive, transitive closure of $R^{\mathcal{M}}$. Note that $(I(u), I(v))$ is in the reflexive, transitive closure of $R^{\mathcal{M}}$. Hence $\mathcal{M}_T \vDash_{I'} \neg C_3$.

In all cases, we have $\mathcal{M}_T \vDash_I \exists x \exists y \exists z(\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$. □

- Reachability is in fact expressible in existential second-order logic.

Reachability in Universal Second-Order Logic III

- Given an existential second-order logic formula ϕ, whether there is an existential second-order logic formula ψ such that ψ and $\neg\phi$ are equivalent is an open problem.

Second- and Higher-Order Logic

- If we allow both quantifiers in a formula, we get second-order logic.
  - For instance, $\exists P \forall Q (\forall x \forall y (Q(x,y) \implies Q(y,x)) \implies \forall u \forall v (Q(u,v) \implies P(u,v)))$ is a second-order logic sentence.
- Furthermore, if we allow quantifiers over relations of relations, we get third-order logic.
- Designing higher-order logic need be careful.
  - Nice properties such as compactness and completeness often fail.
  - Soundness theorem can also fail!
    - ★ Consider $A \triangleq \{x : x \notin x\}$.
- Many theorem provers (Coq, Isabelle, HOL etc) are in fact based on higher-order logics.

Outline

The Coq Proof Assistant

- Coq is a proof assistant which checks every proof steps.
- It has been developed by *Institut national de recherche en informatique et en automatique* (INRIA) at France since 1984.
- It is used to check the proofs of the four color theorem (September 2004) and Feit-Thompson theorem (September 2012).
- It is also used in the CompCert project to formally verify an optimizing C compiler for PowerPC, ARM, and 32-bit x86 processors (2005).
- Coq is available on various platforms.
- The contents of this lecture are borrowed from Coq Tutorial.

Using Coq

- We start up and exit Coq as follows.

```
$ coqtop
Welcome to Coq 8.3pl4 (April 2012)

Coq < Quit .
$
```

Prop, Set, and Type

- A sort classifies specifications.
  - a logical proposition has the sort `Prop`;
  - a mathematical collection has the sort `Set`; and
  - an abstract type has the sort `Type`.
- Every Coq expression has a sort.

```
Coq < Check False .
False
     : Prop

Coq < Check nat .
nat
     : Set

Coq < Check O .
O
     : nat
```

Basic Proof Tactics I

- Let us do some simple proofs.
- We first set up our context .

```
Coq < Section Simple .

Coq < Hypothesis P Q : Prop .
P is assumed
Q is assumed
```

- In this code, we start a section called Simple.
- We also make two hypotheses. Both P and Q are logical propositions.

Basic Proof Tactics II

- We first show $P \implies P$.

```
Coq < Lemma one_line : P -> P .
1 subgoal

P : Prop
Q : Prop
============================
P -> P
```

- We declare a lemma called one_line.
- Coq asks us to show $P \implies P$ from the hypotheses P and Q.

Basic Proof Tactics III

- The tactic `intros` introduces new hypotheses with the given name.

```
one_line < intros HP .
1 subgoal

P : Prop
Q : Prop
HP : P
============================
P
```

- How does `intros` compare to the $\implies i$ rule?

Basic Proof Tactics IV

- The tactic `exact` uses the named hypothesis.

```
one_line < exact HP .
Proof completed.
```

- The command `Qed` finishes up the lemma.

```
one_line < Qed .
intros HP.
exact HP.

one_line is defined
```

Basic Proof Tactics V

- We can check our new lemma and print its proof.

```
Coq < Check one_line .
one_line
     : P -> P

Coq < Print one_line .
one_line = fun HP : P => HP
     : P -> P
```

- Observe how our proof is represented in Coq.

Basic Proof Tactics VI

- Tactics start with lowercase letters such as `intros` and `exact`.
  - We use tactics to construct formal proofs.
- Commands on the other hand start with uppercase letters such as `Quit`, `Section`, `Lemma`, `Qed`, `Print`.
  - We use commands to operate Coq.

Basic Proof Tactics VII

- Let us prove $P \implies (P \implies Q) \implies Q$.

```
Coq < Lemma MP : P -> (P -> Q) -> Q .
1 subgoal

P : Prop
Q : Prop
============================
P -> (P -> Q) -> Q

MP < intros HP HI .
1 subgoal

P : Prop
Q : Prop
HP : P
HI : P -> Q
============================
Q
```

Basic Proof Tactics VIII

- The tactic apply matches the conclusion with the named hypothesis and lists unresolved conditions.

```
MP < apply HI .
1 subgoal

P : Prop
Q : Prop
HP : P
HI : P -> Q
============================
P

MP < exact HP .
Proof completed.
```

- How does apply compare to $\Longrightarrow e$?

Basic Proof Tactics IX

- Let us finish up the lemma and see the proof term.

```
MP < Qed .
intros HP HI.
apply HI.
exact HP.

MP is defined

Coq < Print MP .
MP = fun (HP : P) (HI : P -> Q) => HI HP
      : P -> (P -> Q) -> Q
```

Basic Proof Tactics X

- Let us prove $P \wedge Q \implies Q \wedge P$.

```
    Coq < Lemma conj_comm : P /\ Q -> Q /\ P .
    1 subgoal

    P : Prop
    Q : Prop
    ============================
    P /\ Q -> Q /\ P

conj_comm < intros conj .
    1 subgoal

    P : Prop
    Q : Prop
    conj : P /\ Q
    ============================
    Q /\ P
```

Basic Proof Tactics XI

- The tactic `elim` eliminates a named hypothesis.

```
conj_comm < elim conj .
1 subgoal

P : Prop
Q : Prop
conj : P /\ Q
============================
P -> Q -> Q /\ P
```

- Observe that $P \wedge Q$ is decomposed into P and Q.
- How does `elim` compare to $\wedge e_1$ and $\wedge e_2$?

Basic Proof Tactics XII

- We introduce two more hypotheses *HP* and *HQ*.

```
conj_comm < intros HP HQ .
1 subgoal

P : Prop
Q : Prop
conj : P /\ Q
HP : P
HQ : Q
============================
Q /\ P
```

- Now we can use the hypotheses *HP* and *HQ*.

Basic Proof Tactics XIII

- The tactic `split` splits a conjunction into two.

```
conj_comm < split .
2 subgoals

P : Prop
Q : Prop
conj : P /\ Q
HP : P
HQ : Q
============================
Q

subgoal 2 is:
P
```

- How does `split` compare to $\wedge i$?

Basic Proof Tactics XIV

- We use hypotheses to prove the lemma.

```
conj_comm < exact HQ .
1 subgoal

P : Prop
Q : Prop
conj : P /\ Q
HP : P
HQ : Q
============================
P

conj_comm < exact HP .
Proof completed.
```

Basic Proof Tactics XV

- Let us finish up the lemma and see its proof term.

```
conj_comm < Qed .
intros conj.
elim conj.
intros HP HQ.
split.
 exact HQ.

 exact HP.

conj_comm is defined

Coq < Print conj_comm .
conj_comm =
fun conj0 : P /\ Q =>
    and_ind (fun (HP : P) (HQ : Q) =>
                conj HQ HP) conj0
     : P /\ Q -> Q /\ P
```

Basic Proof Tactics XVI

- Let us try to prove $P \vee Q \implies Q \vee P$.

```
Coq < Lemma disj_comm : P \/ Q -> Q \/ P .
1 subgoal

P : Prop
Q : Prop
============================
P \/ Q -> Q \/ P

disj_comm < intros disj .
1 subgoal

P : Prop
Q : Prop
disj : P \/ Q
============================
Q \/ P
```

Basic Proof Tactics XVII

- We eliminate the hypothesis `disj`.

```
disj_comm < elim disj .
2 subgoals

P : Prop
Q : Prop
disj : P \/ Q
============================
P -> Q \/ P

subgoal 2 is:
Q -> Q \/ P
```

- How does `elim` compare to $\vee e$?

Basic Proof Tactics XVIII

- We next introduce a new hypothesis P.

```
disj_comm < intros HP .
2 subgoals

P : Prop
Q : Prop
disj : P \/ Q
HP : P
============================
Q \/ P

subgoal 2 is:
Q -> Q \/ P
```

Basic Proof Tactics XIX

- The tactic `right` selects the left operand in a disjunction.

```
disj_comm < right .
2 subgoals

P : Prop
Q : Prop
disj : P \/ Q
HP : P
============================
P

subgoal 2 is:
Q -> Q \/ P
```

- How does `right` compare to $\vee i_2$?

Basic Proof Tactics XX

- The tactic `assumption` searches an exact hypothesis for the conclusion.

```
disj_comm < assumption .
1 subgoal

P : Prop
Q : Prop
disj : P \/ Q
============================
Q -> Q \/ P
```

- We can combine a sequence of tactics by semicolon (`;`).

```
disj_comm < intros HQ; left; assumption .
Proof completed.
```

Basic Proof Tactics XXI

- We finish up the lemma and print our proof.

```
disj_comm < Qed .
intros disj.
elim disj.
 intros HP.
 right.
 assumption.

 intros HQ; left; assumption.

disj_comm is defined

Coq < Print disj_comm .
disj_comm =
fun disj : P \/ Q =>
    or_ind (fun HP : P => or_intror Q HP)
           (fun HQ : Q => or_introl P HQ) disj
     : P \/ Q -> Q \/ P
```

Basic Proof Tactics XXII

- Let us prove a lemma about double negation: $P \implies \neg\neg P$.

```
Coq < Lemma PNNP : P -> ~~P .
1 subgoal

P : Prop
Q : Prop
============================
P -> ~ ~ P

PNNP < intros HP .
1 subgoal

P : Prop
Q : Prop
HP : P
============================
~ ~ P
```

Basic Proof Tactics XXIII

- In Coq, $\neg P$ is a shorthand for $P \implies \bot$.
- We use `red` to expand a toplevel shorthand.

```
PNNP < red .
1 subgoal

P : Prop
Q : Prop
HP : P
============================
~ P -> False
```

- We introduce another hypothesis $\neg P$.

```
PNNP < intros HNP .
1 subgoal

P : Prop
Q : Prop
HP : P
HNP : ~ P
============================
False
```

- How does this `intros` compare to $\neg i$?

Basic Proof Tactics XXV

- We have P and $\neg P$. The tactic absurd P exploits the contraction.

```
PNNP < absurd P .
2 subgoals

P : Prop
Q : Prop
HP : P
HNP : ~ P
============================
~ P

subgoal 2 is:
P
```

- How does absurd compare to $\neg e$?

Basic Proof Tactics XXVI

- The tactic `trivial` performs a simple proof search.

```
PNNP < trivial .
1 subgoal

P : Prop
Q : Prop
HP : P
HNP : ~ P
============================
P

PNNP < trivial .
Proof completed.
```

Basic Proof Tactics XXVII

- Let us finish up the lemma, conclude the section, and check it.

```
PNNP < Qed .
intros HP.
red.
intros HNP.
absurd P.
 trivial.

 trivial.

PNNP is defined

Coq < End Simple .

Coq < Check PNNP .
PNNP
     : forall P : Prop, P -> ~ ~ P
```

- Note the hypothesis P is generalized after closing the section.

Basic Proof Tactics XXVIII

- Coq actually provides a complete tactic tauto.

```
Coq < Hypotheses P Q R S : Prop .
P is assumed
Q is assumed
R is assumed
S is assumed
Coq < Hypothesis H0 : (P /\ Q) -> R .
H0 is assumed
Coq < Hypothesis H1 : R -> S .
H1 is assumed
Coq < Hypothesis H2 : Q /\ ~S .
H2 is assumed
Coq < Lemma homework : ~P .
1 subgoal

P : Prop
Q : Prop
R : Prop
S : Prop
H0 : P /\ Q -> R
H1 : R -> S
H2 : Q /\ ~ S
============================
~ P
homework < tauto .
Proof completed.
```

Basic Proof Tactics XXIX

- COQ in fact uses intuitionistic logic.

```
Coq < Goal forall P : Prop, P \/ ~P .
1 subgoal

============================
forall P : Prop, P \/ ~ P

Unnamed_thm < tauto .
Toplevel input, characters 0-5:
> tauto .
> ^^^^^
Error: tauto failed.
```

  - `Goal` declares an unnamed lemma.

- To do classical logic, add

```
Coq < Require Import Classical .

Coq < Check classic .
classic
     : forall P : Prop, P \/ ~ P
```

More Proof Tactics I

- Let us set up a section for predicate logic.

```
Coq < Section Easy .

Coq < Hypothesis D : Set .
D is assumed

Coq < Hypothesis R : D -> D -> Prop .
R is assumed
```

- In a new section, we declare a set D and a binary predicate symbol R.
- Let us set up a subsection where R is symmetric and transitive.

```
Coq < Section R_sym_trans .

Coq < Hypothesis R_symmetric :
          forall x y : D, R x y -> R y x .
R_symmetric is assumed

Coq < Hypothesis R_transitive :
          forall x y z : D, R x y -> R y z -> R x z .
R_transitive is assumed
```

More Proof Tactics II

- Let us prove $\forall x \in D (\exists y \in D, (Rxy) \implies Rxx)$.

```
Coq < Lemma refl_if :
                forall x : D, (exists y, R x y) -> R x x .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
============================
forall x : D, (exists y : D, R x y) -> R x x
```

- Our predicate logic formula is written as `forall x :  D, (exists y, R x y) -> R x x .`
- Observe that we did not specify $y \in D$ but COQ infers it anyway.

More Proof Tactics III

- The tactic `intros` again introduces a new hypothesis.

```
refl_if < intros x .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
============================
(exists y : D, R x y) -> R x x
```

- How does it compare to $\forall i$?

More Proof Tactics IV

- We introduce another hypothesis $\exists y \in D(Rxy)$.

```
refl_if < intros Ey .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
Ey : exists y : D, R x y
============================
R x x
```

- This is simply $\implies i$.

More Proof Tactics V

- Let us eliminate $\exists y \in D(Rxy)$.

```
refl_if < elim Ey .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
Ey : exists y : D, R x y
============================
forall x0 : D, R x x0 -> R x x
```

- How does `elim` compare to $\exists e$?

More Proof Tactics VI

- We get the instance of $\exists y \in D(Rxy)$ by `intros`.

```
refl_if < intros y Rxy .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
Ey : exists y : D, R x y
y : D
Rxy : R x y
============================
R x x
```

- Now `elim` and `intros` look really like $\exists e$.

More Proof Tactics VII

- We apply the hypothesis `R_transitive`.

```
refl_if < apply R_transitive with y .
2 subgoals

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
Ey : exists y : D, R x y
y : D
Rxy : R x y
============================
R x y

subgoal 2 is:
R y x
```

- Note that we need to give the hint `y`.
- How does `apply` compare to $\forall e$?

More Proof Tactics VIII

- The first subgoal is trivial.

```
refl_if < trivial .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
Ey : exists y : D, R x y
y : D
Rxy : R x y
============================
R y x
```

More Proof Tactics IX

- For the other subgoal, we apply $\forall xy \in D(Rxy \implies Ryx)$.

```
refl_if < apply R_symmetric .
1 subgoal

D : Set
R : D -> D -> Prop
R_symmetric : forall x y : D, R x y -> R y x
R_transitive : forall x y z : D, R x y -> R y z -> R x z
x : D
Ey : exists y : D, R x y
y : D
Rxy : R x y
============================
R x y
```

- Now the goal is trivial.

```
refl_if < trivial .
Proof completed.
```

More Proof Tactics X

- Let us finish up the lemma and see the proof term.

```
refl_if < Qed .
intros x.
intros Ey.
elim Ey.
intros y Rxy.
apply R_transitive with y.
 trivial.

 apply R_symmetric.
 trivial.

refl_if is defined

Coq < Print refl_if .
refl_if =
fun (x : D) (Ey : exists y : D, R x y) =>
ex_ind
(fun (y : D) (Rxy : R x y) =>
     R_transitive x y x Rxy (R_symmetric x y Rxy)) Ey
   : forall x : D, (exists y : D, R x y) -> R x x
```

Smullyan's Drinkers' Paradox I

- We will prove Smullyan's drinkers' paradox:
  "in any non-empty bar, there is a person such that she drinks then everyone drinks."
- Let us set up the context.

```
Coq < Section DrinkersParadox .

Coq < Require Import Classical .

Coq < Hypothesis bar : Set .
bar is assumed

Coq < Hypothesis Joe : bar .
Joe is assumed

Coq < Hypothesis drinks : bar -> Prop .
drinks is assumed
```

- Note that Joe is in the bar.

- Here is what we want to prove.

```
Coq < Lemma drinker : exists x : bar, drinks x ->
                                forall y : bar, drinks y .
1 subgoal

bar : Set
Joe : bar
drinks : bar -> Prop
============================
exists x : bar, drinks x -> forall y : bar, drinks y
```

Smullyan's Drinkers' Paradox III

- By LEM, we have $(\exists x \in bar(\neg drinks\ x)) \vee \neg(\exists x \in bar(\neg drinks\ x))$.
- We consider the two cases.

```
drinker < Check (classic (exists x : bar, ~ drinks x)) .
classic (exists x : bar, ~ drinks x)
     : (exists x : bar, ~ drinks x) \/
       ~ (exists x : bar, ~ drinks x)

drinker < elim (classic (exists x : bar, ~ drinks x)) .
2 subgoals

bar : Set
Joe : bar
drinks : bar -> Prop
============================
(exists x : bar, ~ drinks x) ->
exists x : bar, drinks x -> forall y : bar, drinks y

subgoal 2 is:
~ (exists x : bar, ~ drinks x) ->
exists x : bar, drinks x -> forall y : bar, drinks y
```

Smullyan's Drinkers' Paradox IV

- We introduce the hypothesis non_drinker.

```
drinker < intros non_drinker .
2 subgoals

bar : Set
Joe : bar
drinks : bar -> Prop
non_drinker : exists x : bar , ~ drinks x
============================
exists x : bar , drinks x -> forall y : bar , drinks y

subgoal 2 is:
~ (exists x : bar , ~ drinks x) ->
exists x : bar , drinks x -> forall y : bar , drinks y
```

- We eliminate non_drinker and obtain an instance.

```
drinker < elim non_drinker; intros Jane Jane_non_drinker .
2 subgoals

bar : Set
Joe : bar
drinks : bar -> Prop
non_drinker : exists x : bar, ~ drinks x
Jane : bar
Jane_non_drinker : ~ drinks Jane
============================
exists x : bar, drinks x -> forall y : bar, drinks y

subgoal 2 is:
~ (exists x : bar, ~ drinks x) ->
exists x : bar, drinks x -> forall y : bar, drinks y
```

Smullyan's Drinkers' Paradox VI

- The tactic exists uses a term as a witness to an existential formula.

```
drinker < exists Jane .
2 subgoals

bar : Set
Joe : bar
drinks : bar -> Prop
non_drinker : exists x : bar, ~ drinks x
Jane : bar
Jane_non_drinker : ~ drinks Jane
==============================
drinks Jane -> forall y : bar, drinks y

subgoal 2 is:
~ (exists x : bar, ~ drinks x) ->
exists x : bar, drinks x -> forall y : bar, drinks y
```

- How does exists compare to $\exists i$?

Smullyan's Drinkers' Paradox VII

- Observe that we have a contradiction.
- The tactic tauto will do.

```
drinker < tauto .
1 subgoal

bar : Set
Joe : bar
drinks : bar -> Prop
============================
~ (exists x : bar, ~ drinks x) ->
exists x : bar, drinks x -> forall y : bar, drinks y
```

Smullyan's Drinkers' Paradox VIII

- We introduce a hypothesis for the other subgoal.

```
drinker < intros no_non_drinker .
1 subgoal

bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
============================
exists x : bar, drinks x -> forall y : bar, drinks y
```

Smullyan's Drinkers' Paradox IX

- Joe is our witness.

```
drinker < exists Joe .
1 subgoal
bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
============================
drinks Joe -> forall y : bar, drinks y
```

- We introduce more hypotheses.

```
drinker < intros Joe_drinker y .
1 subgoal
bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
Joe_drinker : drinks Joe
y: bar
============================
drinks y
```

Smullyan's Drinkers' Paradox X

- For $y \in bar$, we have $drinks\ y \lor \neg drinks\ y$ by LEM.

```
drinker < elim (classic (drinks y)) .
2 subgoals

bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
Joe_drinker : drinks Joe
y : bar
============================
drinks y -> drinks y

subgoal 2 is:
~ drinks y -> drinks y
```

Smullyan's Drinkers' Paradox XI

- The first subgoal is easy.

```
drinker < tauto .
1 subgoal

bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
Joe_drinker : drinks Joe
y : bar
============================
~ drinks y -> drinks y
```

Smullyan's Drinkers' Paradox XII

- We introduce a hypothesis that y does not drink.

```
drinker < intros y_non_drinker .
1 subgoal

bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
Joe_drinker : drinks Joe
y : bar
y_non_drinker : ~ drinks y
============================
drinks y
```

Smullyan's Drinkers' Paradox XIII

- This is contradictory to `no_non_drinker`.

```
drinker < absurd (exists x, ~ drinks x) .
2 subgoals

bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
Joe_drinker : drinks Joe
y : bar
y_non_drinker : ~ drinks y
============================
~ (exists x : bar, ~ drinks x)

subgoal 2 is:
exists x : bar, ~ drinks x
```

Smullyan's Drinkers' Paradox XIV

- Again, the first subgoal is trivial.
- The second subgoal has a witness y.

```
drinker < trivial .
1 subgoal

bar : Set
Joe : bar
drinks : bar -> Prop
no_non_drinker : ~ (exists x : bar, ~ drinks x)
Joe_drinker : drinks Joe
y : bar
y_non_drinker : ~ drinks y
============================
exists x : bar, ~ drinks x

drinker < exists y; trivial .
Proof completed.
```

Smullyan's Drinkers' Paradox XV

- Let us finish up the lemma and see its proof term.

```
drinker < Qed .
(* proof script skipped *)

Coq < Print drinker .
drinker =
or_ind
  (fun non_drinker : exists x : bar, ~ drinks x =>
   ex_ind
     (fun (Jane : bar) (Jane_non_drinker : ~ drinks Jane) =>
      ex_intro (fun x : bar => drinks x -> forall y : bar, drinks y) Jane
        (fun H : drinks Jane =>
         let H0 := Jane_non_drinker H in
         False_ind (forall y : bar, drinks y) H0)) non_drinker)
  (fun no_non_drinker : ~ (exists x : bar, ~ drinks x) =>
   ex_intro (fun x : bar => drinks x -> forall y : bar, drinks y) Joe
     (fun (_ : drinks Joe) (y : bar) =>
      or_ind (fun H : drinks y => H)
        (fun y_non_drinker : ~ drinks y =>
         False_ind (drinks y)
           (let H := ex_intro (fun x : bar => ~ drinks x) y y_non_drinker in
            (let H0 := no_non_drinker in
             fun H1 : exists x : bar, ~ drinks x => H0 H1) H))
        (classic (drinks y)))) (classic (exists x : bar, ~ drinks x))
     : exists x : bar, drinks x -> forall y : bar, drinks y
```

Where to go?

- Proof assistants are used to check long proofs in mathematics and logic.
  - Four color theorem, Feit-Thompson theorem, incompleteness theorem.
- We only discuss elements of predicate logic.
- Lots of interesting topics are missing. For instance,
  - Soundness and competeness theorems of natural deduction for predicate logic;
  - Gödel's incompleteness theorem;
  - Number theory, real analysis COQ libraries.
- Many resources are available for learning COQ.
  - Short NTU summer courses (FLOLAC).
  - "*Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*", Bertot and Castéran.