Outline

Propositional Logic and Infinite Domains

Consider simple propositions:

1. The successor of 0 is greater than 0;
2. The successor of 1 is greater than 1;
3. The successor of 2 is greater than 2;
4. The successor of 3 is greater than 3;
5. The successor of 4 is greater than 4;
6. ...

We can use them in propositional logic. But how can we express the property

- The successor of every natural number is greater than this number?

To express it we need a conjunction of an infinite number of propositions.

First order logic

In first order logic we can express properties of the form for all ... and there exists ... using quantifiers.
For example, to express the successor property we can

- introduce a function symbol *succ* to represent the successor function, so that $succ(x)$ denotes the successor of x;
- introduce a predicate symbol $>$ to represent the order on numbers, so that $x > y$ denotes that x is greater than y;
- use a quantifier $\forall$ to express that the successor of every number is greater than this number as $(\forall x)(succ(x) > x)$.

Syntax: the Language

Signature Σ: a set of

- constants;
- function symbols;
- predicate symbols.

Each function symbol and predicate symbol has an associated arity (the number of arguments).
In addition to elements of the signature, the language will use a countably infinite set of variables.
Example: $succ(x) > x$, here

- x is a variable;
- $succ$ is a function symbol of arity 1 (unary function symbol);
- $>$ is a predicate symbol of arity 2 (binary predicate symbol).

Predicate symbols are sometime called relation symbols.

Syntax: Terms

For convenience we fix a signature.

Term:

- every variable is a term;
- every constant is a term;
- if f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

Examples:

- x;
- 0;
- $succ(succ(x))$;
- $x + y$ (here the binary function symbol $+$ is written in the infix notation).

Two notations

Prolog notation:

- Variables start with upper-case letters: `X`, `Man`.
- Constants start with lower-case letters: `x`, `man`.

Math notation:

- Variables: x, y, z, u, v, w.
- Constants: a, b, c, d, e.

First-Order Formula

- If p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula, also called an atomic formula, or simply atom.
- $\top$nd $\bot$ are formulas.
- If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \land \ldots \land A_n)$ and $(A_1 \lor \ldots \lor A_n)$ are formulas.
- If A is a formula, then $(\neg A)$ is a formula.
- If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.
- If A is a formula and x is a variable then $(\forall x)A$ and $(\exists x)A$ are formulas.

Quantifiers

$(\forall x)A$: A holds for all x.
$(\exists x)A$: A holds for some x or there exists some x such that A.
$\forall$: universal quantifier.
$\exists$: existential quantifier.

Free and Bound Variables

Variable Binding:

- x is bound in $\forall x\ F$ or $\exists x\ F$.
- F is the scope of x
- A variable which is not bound is free.

A formula with no free variables is called closed.

Semantics: Structure

Structure $M = (D, R, F, C)$:

- domain D (non-empty)
- R: assign k-ary relation P^M on D to each k-ary predicate symbol P of L;
- F: assign k-ary function f^M on D to each k-ary function symbol f of L;
- C: assign element a^M from D to each constant symbol a of L.

Value assignment s over M: maps variables to domain elements, that is, $s(x) \in D$.

Values for Terms

t is given value $t^{M,s} \in D$:

Term	Value in D
constant a	$a^{M,s} = a^M$
variable x	$x^{M,s} = s(x)$
n-ary function f	$f(t_1, t_2, \ldots, t_n)^{M,s} = f^M(t_1^{M,s}, t_2^{M,s}, \ldots, t_n^{M,s})$ ($t_1, \ldots, t_n$ are terms)

Notation

Define

$$s[x \leftarrow d](y) \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} s(x), & \text{if } x \neq y; \\ d, & \text{if } x = y. \end{cases}$$

Semantics: Truth

Truth values for formulae: $v_{M,s}(A) \in \{1, 0\}$

Logical Symbol	Truth Value
constant $\top$	$v_{M,s}(\top) = 1$
constant $\bot$	$v_{M,s}(\bot) = 0$
predicate	$v_{M,s}(P(t_1, \ldots, t_n)) = 1$ iff $(t_1^{M,s}, \ldots, t_n^{M,s}) \in P^M$
connective (e.g)	$v_{M,s}(F \wedge G) = 1$ iff $v_{M,s}(F) = 1$ and $v_{M,s}(G) = 1$
quantifier $\forall$	$v_{M,s}(\forall x F) = 1$ iff for all $d \in D$, $v_{M,s[x \leftarrow d]}(F) = 1$
quantifier $\exists$	$v_{M,s}(\exists x F) = 1$ iff for some $d \in D$, $v_{M,s[x \leftarrow d]}(F) = 1$

If $v_{M,s}(F) = 1$, write $M, s \models F$

Satisfiability and validity

If A is closed, then $v_{M,s}(A)$ is independent of s; so we write $M \models A$ and say that is true in M.

- If a formula A is true in M we say that M satisfies A and that M is a model of A, denoted by $M \models A$.
- A is satisfiable (valid) if it is true in some (every) structure.
- Two formulas A and B are called equivalent, denoted $A \equiv B$ if they have the same models.

Example

$A = \forall x \forall y (q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))$

Take the tructure $M = (people, \{q \mapsto ancestor, p \mapsto parent\}, \emptyset, \emptyset)$ and any value assignment s:

- $v_{M,s}(\forall x \forall y (q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))))) = 1$ iff
- for all $d \in D$, $v_{M,s[x \leftarrow d]}(\forall y (q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))))) = 1$ iff
- for all $d \in D$, for all $d' \in D$, $v_{M,s[x \leftarrow d][y \leftarrow d']}(q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y)))) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $v_{M,s[x \mapsto d][y \mapsto d']}(q(x,y)) = 1$ then $v_{M,s[x \mapsto d][y \mapsto d']}(p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $(d, d') \in ancestor$ then either $v_{M,s[x \mapsto d][y \mapsto d']}(p(x,y))$ or $v_{M,s[x \mapsto d][y \mapsto d']}(\exists z (p(x,z) \wedge q(z,y)))$ iff
- for all $d \in D$, for all $d' \in D$, if $(d, d') \in ancestor$ then either $(d, d') \in parent$ or there exists a $d'' \in D$, such that $v_{M,s[x \mapsto d][y \mapsto d'][z \mapsto d'']}(p(x,z) \wedge q(z,y)) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $(d, d') \in ancestor$ then either $(d, d') \in parent$ or there exists a $d'' \in D$, such that $v_{M,s[x \mapsto d][y \mapsto d'][z \mapsto d'']}(p(x,z)) = 1$ and $v_{M,s[x \mapsto d][y \mapsto d'][z \mapsto d'']}(q(z,y)) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $(d, d') \in ancestor$ then either $(d, d') \in parent$ or there exists a $d'' \in D$, such that $(d, d'') \in parent$ and

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
- Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
- Unit clause: $n = 1$.

When we consider clauses we assume that the order of literals in them is irrelevant.

Negation Normal Form

A formula A is in negation normal form, or simply NNF, if it is either $\top$, or $\bot$, or is built from literals using only $\land$, $\lor$, $\forall$ and $\exists$.

A formula B is called a negation normal form of a formula A if B is equivalent to A and B is in negation normal form.

Negation Normal Form

A formula A is in negation normal form, or simply NNF, if it is either $\top$, or $\bot$, or is built from literals using only $\wedge$, $\vee$, $\forall$ and $\exists$.

A formula B is called a negation normal form of a formula A if B is equivalent to A and B is in negation normal form.

NNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg(\forall x)A & \Rightarrow & (\exists x)\neg A, \\
\neg(\exists x)A & \Rightarrow & (\forall x)\neg A, \\
\neg\neg A & \Rightarrow & A
\end{array}
$$

Rectified formulas

Rectified formula F:

- no variable appears both free and bound in F;
- for every variable x, the formula F contains at most one occurrence of quantifiers $\forall x$ or $\exists x$.

Any formula can be transformed into a rectified formula by renaming bound variables.

Rectification: Example

$$
\begin{aligned}
&p(x) \to \exists x(p(x) \land \forall x(p(x) \lor r \to \neg p(x))) \Rightarrow \\
&p(x) \to \exists x_1(p(x_1) \land \forall x(p(x) \lor r \to \neg p(x))) \Rightarrow \\
&p(x) \to \exists x_1(p(x_1) \land \forall x_2(p(x_2) \lor r \to \neg p(x_2)))
\end{aligned}
$$

Skolemisation: Choice Functions

We would like to get rid of existential quantifiers using choice functions, or witness functions.
Consider an example. We know that every tree has a root:

$$\forall x(tree(x) \rightarrow \exists y(root(y, x))). \qquad (*)$$

Then we can introduce a function, say *rootof* that gives the root of a tree and write

$$\forall x(tree(x) \rightarrow root(rootof(x), x)). \qquad (**)$$

Note that $(*)$ is a logical consequence of $(**)$.

Skolemisation: Choice Functions

We would like to get rid of existential quantifiers using choice functions, or witness functions.
Consider an example. We know that every tree has a root:

$$\forall x(tree(x) \to \exists y(root(y, x))). \qquad (*)$$

Then we can introduce a function, say *rootof* that gives the root of a tree and write

$$\forall x(tree(x) \to root(rootof(x), x)). \qquad (**)$$

Note that $(*)$ is a logical consequence of $(**)$.

Skolemisation

Let A be a closed rectified formula in NNF and $(\exists x)B$ be a subformula of A. Let $(\forall x_1), \ldots, (\forall x_n)$ be all universal quantifiers such that $(\exists x)B$ is in the scope of these quantifiers. Then:

1. remove $(\exists x)$ from A.
2. replace x everywhere in A by $f(x_1, \ldots, x_n)$, where f is a new function symbol.

Skolemisation does not preserve equivalence but preserves satisfiability.

CNF Transformation

Take a first-order formula F.

1. transform it into NNF;
2. rectify it;
3. skolemise it;
4. remove all universal quantifiers;
5. transform to CNF the same way as propositional formulas.

CNF Transformation

Universal closure of a formula A is a formula

$$(\forall x_1)\dots(\forall x_n)A,$$

denoted by $\forall A$, where $x_1,\dots,x_n$ are all free variables of A.

CNF transformation transforms a closed formula F into a set of clauses $C_1,\dots,C_n$ such that F is satisfiable if and only if so is the set of formulas $\forall C_1,\dots,\forall C_n$.

CNF Transformation

Universal closure of a formula A is a formula

$$(\forall x_1)\ldots(\forall x_n)A,$$

denoted by $\forall A$, where $x_1, \ldots, x_n$ are all free variables of A.

CNF transformation transforms a closed formula F into a set of clauses $C_1, \ldots, C_n$ such that F is satisfiable if and only if so is the set of formulas $\forall C_1, \ldots, \forall C_n$.