Background Review

1 Propositional Logic Informally

Propositional logic is the core part of most logical formalisms (such as first-order and higher-order logic). It also describes well digital circuits that are the basis of most modern computing devices.

1.1 Importance of Propositional Logic

Many search problems can be encoded using propositional formulas

Checking equivalence of logical combinatorial circuits in hardware directly reduces to SAT

Can encode integer arithmetic with fixed bitwidth (e.g. operations on 16-bit integers)

An important class of logics can be obtained by giving meaning to propositional variables

1.2 Propositional Formulas

Propositional logic studies propositional formulas.

Propositional formulas are expressions built from

constants true and false

propositional variables, such as p, q, p_1, q_1,

logical operators ($\wedge, \vee, \neg, \rightarrow, \leftrightarrow$) that combine propositional formulas into larger propositional formulas (we define their meaning below)

Analogy with mathematical expressions: Just like a mathematical expression such as $2 \cdot x + 3 \cdot y$ specifies a binary function from values of x and y to the value $2x + 3y$, a propositional formula such as $p \wedge (\neg q)$ specifies a function from the values of p and q to the value $p \wedge (\neg q)$.

1.3 Definitions of Logical Operations (Logical Connectives)

We specify a binary logical operation by a truth table: for each combination of arguments we indicate the result of the operation. For binary operation, the rows indicate the first argument, the columns indicate the second argument.

1.3.1 Negation (logical 'not')

p	$\neg p$
$false$	$true$
$true$	$false$

1.3.2 Conjunction (logical 'and')

$\wedge$	$false$	$true$
$false$	$false$	$false$
$true$	$false$	$true$

1.3.3 Disjunction (logical 'or')

$\vee$	$false$	$true$
$false$	$false$	$true$
$true$	$true$	$true$

1.3.4 Implication ('if')

$\rightarrow$	$false$	$true$
$false$	$true$	$true$
$true$	$false$	$true$

Check validity of these implications:
$(1 = 2)\ \rightarrow$ (7 is an even number)
$(1 = 1)\ \rightarrow$ (7 is an even number)
$(1 = 2)\ \rightarrow$ ($\sqrt{2}$ is a rational number)
Logical Eqivalence ('if and only if')

$\leftrightarrow$	$false$	$true$
$false$	$true$	$false$
$true$	$false$	$true$

1.4 Evaluating Propositional Formulas

If we know values of variables in the formula, we can compute its value.
Let us evaluate formula

$$((p \rightarrow q) \wedge\ ((\neg p) \rightarrow r)) \leftrightarrow ((p \wedge q)\ \vee\ ((\neg p) \wedge r))$$

for all values of its parameters. Let us draw formula as a tree. We introduce a column for each tree node. We obtain the value of a tree node by looking at the value of its children and applying the truth table for the operation in the node. The root gives the truth value of the formula.

1.5 Validity, Satisfiability of Logical Formulas

Formula is valid if it evaluates to true for all values of its variables (all columns for formula say true).

Formula is satisfiable if it evaluates to true for some values of its variables (true appears in at least one column).

Formula is unsatisfiable if it evaluates to false for all values of its variables (all columns for formula say false).

1.5.1 Observations

F is unsatisfiable if and only if F is not satisfiable

F is valid if and only if $\neg F$ is unsatisfiable

SAT problem: given a propositional formula F, check whether F is satisfiable.

SAT is a well-known NP-complete problem.

1.6 Propositional Tautologies

Valid propositional formula is called propositional tautology, or tautology for short. Tautologies are basic principles of logical reasoning. They are true regardless of how complex the intended meaning of propositions is, as long as we are sure that each proposition evaluates to true or false.

Here is a small list of tautologies.

$$
\begin{array}{l}
(p \to q) \leftrightarrow ((\neg p) \lor q) \\
(p \leftrightarrow q) \leftrightarrow ((p \to q) \land (q \to p)) \\
(p \land q) \leftrightarrow (q \land p) \\
(p \land (q \land r)) \leftrightarrow ((p \land q) \land r)) \\
(p \lor q) \leftrightarrow (q \lor p) \\
(p \lor (q \lor r)) \leftrightarrow ((p \lor q) \lor r)) \\
(p \land (q \lor r)) \leftrightarrow ((p \land q) \lor (p \land r)) \\
(p \lor (q \land r)) \leftrightarrow ((p \lor q) \land (p \lor r)) \\
\neg(\neg p) \leftrightarrow p \\
p \lor (\neg p) \\
\neg(p \land q) \leftrightarrow (\neg p) \lor (\neg q) \\
\neg(p \lor q) \leftrightarrow (\neg p) \land (\neg q) \\
(p \to (q \to r)) \leftrightarrow ((p \land q) \to r) \\
(p \to (q \land r)) \leftrightarrow ((p \to q) \land (p \to r)) \\
((p \lor q) \to r) \leftrightarrow ((p \to r) \land (q \to r)) \\
((p \to false) \leftrightarrow (\neg p)
\end{array}
$$

Suggest another tautology.

2 Propositional Logic Syntax

Let V be a countable set of propositional variables, denoted by non-terminal V. The context-free grammar of propositional logic formulas $\mathcal{F}$ is the following:

$$F ::= V \mid \mathit{false} \mid \mathit{true} \mid (F \wedge F) \mid (F \vee F) \mid (\neg F) \mid (F \rightarrow F) \mid (F \leftrightarrow F)$$

We denote the set of all propositional formulas given by the above context-free grammar by $\mathcal{F}$. Each propositional formula is a finite sequence of symbols, given by the above context-free grammar. The set $\mathcal{F}$ is a countable set: we can order all formulas in this set in a sequence (for example, by writing them down in binary alphabet and sorting the resulting strings alphabetically).

Omitting parentheses:

$\wedge$, $\vee$ associative

priorities, from strongest-binding: $(\neg)$; $(\wedge, \vee)$; $(\rightarrow, \leftrightarrow)$

When in doubt, use parentheses.

Notation: when we write $F_1 \equiv F_2$ this means that F_1 and F_2 are identical formulas (with identical syntax trees). For example, $p \wedge q \equiv p \wedge q$, but it is not the case that $p \wedge q \equiv q \wedge p$.

In Isabelle theorem prover we use this

ASCII notation for Propositional Logic

Usually we work with syntax trees, as in Problem 3 in Homework 1.

FV denotes the set of free variables in the given propositional formula and can be defined recursively as follows:

$$\begin{array}{l}
FV(p) = \{p\}, \text{ for } p \in V \\
FV(\neg F) = FV(F) \\
FV(F_1 \wedge F_2) = FV(F_1) \cup FV(F_2) \\
FV(F_1 \vee F_2) = FV(F_1) \cup FV(F_2) \\
FV(F_1 \rightarrow F_2) = FV(F_1) \cup FV(F_2) \\
FV(F_1 \leftrightarrow F_2) = FV(F_1) \cup FV(F_2)
\end{array}$$

If $FV(F) = \emptyset$, we call F a ground formula.

3 Propositional Logic Semantics

Let $\mathcal{B} = \{\mathit{true}, \mathit{false}\}$.

Interpretation for propositional logic is a function $I : V \rightarrow \mathcal{B}$.

We next define evaluation function:

$$e : F \rightarrow (I \rightarrow \mathcal{B})$$

by recursion on formula syntax tree:

This definition follows one in the formula evaluator in homework01.

We denote $e(F)(I) = \mathit{true}$ by

$$I \models F$$

and denote $e(F)(I) = \mathit{false}$ by

$$I \not\models F$$

3.1 Validity and Satisfiability

Formula is valid iff $\forall I. I \models F$. We write this simply

$$\models F$$

Formula is satisfiable iff $\exists I. I \models F$
Formula is contradictory iff $\forall I. I \not\models F$

3.2 Satisfactory Of Sets Of Formulas

We next introduce sets of formulas as a way of talking about potentially infinite conjunctions of formulas. We will need this when reducing reasoning in first-order logic to reasoning in propositional logic.

We say that interpretation I is a model for a set of formulas S, written $I \models S$, iff for each $F \in S$, $I \models F$. In other words, we view a set of formulas as a (potentially infinite) conjunction; when S is finite then $I \models S$ is the same condition as $I \models \bigwedge_{F \in S} F$.

Example: the set $\{p_1, \neg p_1 \vee p_2, \neg p_2 \vee p_3, \neg p_3 \vee p_4, \ldots\}$ is an infinite satisfiable set. The example of one satisfying interpretation is the interpretation which evaluates all variables to $true$.

Clearly, if $I \models B$ and $A \subseteq B$, then also $I \models A$.

We say that a set of formulas S is satisfiable iff there exists an interpretation I such that $I \models S$.

4 Normal Forms for Propositional Logic

4.1 Negation-Normal Form

In Negation-normal from, negations are only allowed on elementary proposition. Moreover, NNF formulas contain no implication, so the only binary operators are conjunctions and disjunctions. The following rules can be used to turn arbitrary propositional formulas into negation-normal form.

$$\begin{array}{l}\neg(p \wedge q) \leftrightarrow (\neg p) \vee (\neg q) \\ p \leftrightarrow \neg(\neg p) \\ (p \rightarrow q) \leftrightarrow ((\neg p) \vee q) \\ \neg(p \vee q) \leftrightarrow (\neg p) \wedge (\neg q)\end{array}$$

Note that this transformation is linear in the size of the formula. No exponential blow-up.

4.2 Disjunctive Normal Form

Formulas in Disjunctive-normal form look like this: $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_3 \wedge x_4) \vee ...$

More formally $F = \bigvee_{i=1}^{n} D_i$ where $n \geq 0$.

Each D_i is a clause and is defined as $D_i = \bigwedge_{j=1}^{n_i} L_{ij}$.

Each L_{ij} is a literal. It's either an elementary proposition or its negation.

Solving the SAT problem for DNF formulas is in P, but transforming an arbitrary propositional formula to DNF causes an exponential blow-up.

DNF formulas can be easily generated from truth tables. Each row of the truth table that makes the formula true can be written as a clause. Here is an example:

x_1	x_2	F
0	0	0
1	0	1
0	1	1
1	1	0

The corresponding formula in DNF is $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

For a formula over n variables, there are 2^n rows in the truth table. Over n variables, there are 2^{2^n} different (i.e. non-equivalent) formulas.

4.3 Conjunctive Normal Form

Formulas in Conjunctive-normal form look like this: $(x_1 \vee x_2 \vee \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge ...$

It's defined as $F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} L_{ij}$

Like for DNF, L_{ij} are elementary propositions or their negation. The terminology of clauses and literals also applies to CNF.

There is no polynomial-time equivalence preserving transformation to CNF or to DNF.

4.4 Complete Sets of Connectives

If we can express every formula. Examples:

$\{\wedge, \vee, \neg\}$ because every formula has DNF (or CNF)

$$\{\wedge, \neg\}$$

$$\{\vee, \neg\}$$

$$\{\rightarrow, \mathit{false}\}$$

$$\{\bar{\wedge}\}$$

$$\{\bar{\vee}\}$$

4.5 Circuits

Formulas can be represented as abstract syntax tree (AST) where each node is labeled with an operator that applies to the sub-tree(s). If two sub-trees are identical, instead of duplicating the sub-tree in each place where its used, one can make all the references to this sub-tree point to a unique representation of it. This is called a circuit.

The if-then-else primitive, written $ite(p, q, r)$, that yields q whenever p is true and r otherwise, can be encoded with the following propositional logic formula: $(p \wedge q) \vee (\neg p \wedge r)$

For each node of an AST, it is possible to replace it with a fresh variable, provided that a clause is added that makes sure that the fresh variable and the sub-tree it represents are equivalent. Note that this transformation preserve equisatisfiability but not equivalence, because it introduces new variables.

4.6 Satisfiability-Preserving Transformation

There exists a linear transformation from arbitrary formulas to CNF preserving equisatisfiability. The main idea is to use fresh variables as described above. For each node of the AST, a representative (i.e. a fresh variable) will be introduced. We need to add clauses to ensure that a sub-formula and its representative are equivalent. To avoid exponential blow-up, we will not use the sub-formulas' children directly, but their representative when expressing this constraint.

The key transformation steps are:

$$\begin{array}{lcl} F & \rightsquigarrow & (p_i \leftrightarrow (q \wedge r)) \wedge subst(\{q \wedge r \mapsto p_i\}, F) \\ F & \rightsquigarrow & (p_i \leftrightarrow (q \vee r)) \wedge subst(\{q \vee r \mapsto p_i\}, F) \\ F & \rightsquigarrow & (p_i \leftrightarrow (\neg q)) \wedge subst(\{(\neg q) \mapsto p_i\}, F) \end{array}$$

Each equivalence between a representative and the ones from the sub-formulas has to be flatten to conjunctive-normal form. This can be done by splitting the equivalences into two implications. Example:

$p \leftrightarrow q_1 \wedge q_2$ becomes $(\neg p \vee q_1) \wedge (\neg p \vee q_2) \wedge (\neg q_1 \vee \neg q_2 \vee p)$

5 Predicate Logic Informally

5.1 First Order Logic

First-order logic is a very powerful notation that extends propositional logic. Here we give only an informal overview of first-order logic, which should be enough for you to understand the meaning of first-order logic formulas based on mathematics and natural language. In future lectures we study first-order logic as a formal system and discuss algorithmic questions in more detail.

On top of propositional operations, first-order logic adds:

1. constructs to represent the structure of propositions:

   (a) equality (=)

   (b) predicate symbols (P, Q,)
   (c) function symbols (f, g,)
   (d) first-order variables (x, y,) denoting entities in some domain D

2. quantifiers forall ($\forall$), exists ($\exists$)

5.2 Uses of first-order logic

1. precisely describe arbitrary mathematical statements (theorems, conjectures, properties)
2. specify program properties
3. represent the meaning of programs
4. represent knowledge about the world (e.g. knowledge bases, semantic web)

5.2.1 Examples

1. An ancestor of my ancestor is also my ancestor: $\forall x.\forall y.\ ((ancestor(x, I) \land ancestor(y, x)) \rightarrow ancestor(y, I))$
2. Grandparent is the parent of a parent: $\forall x.\forall y.\ (grandparent(x, y) \leftrightarrow (\exists z.parent(x, z) \land parent(z, y)))$
3. Property P holds for infinitely many natural numbers: $\forall n.\exists m.\ m > n \land P(m)$
4. f is continuous in point x_0: $\forall \varepsilon > 0.\exists \delta > 0.\forall x.\ |x - x_0| < \delta \rightarrow |f(x) - f(x_0)| < \varepsilon$
5. first k array elements are strictly positive, remaining elements are zero: $\forall i.((0 \leq i \land i < k \rightarrow a(i) > 0) \land ((k \leq i \land i < N) \rightarrow a(i) = 0)$

5.3 First-Order Logic Formulas

We build first-order logic formulas by starting from atomic formulas (defined below) and applying propositional operators and quantifiers:

1. atomic formulas are first-order logic formulas
2. if P and Q are first-order formulas, so are $\neg P$, $P \land Q$, $P \lor Q$, $P \rightarrow Q$, $P \leftrightarrow Q$
3. if P is a first-order formula, so is $\forall x.P$ and $\exists x.P$

For a given value of variables, each formula evaluates to true or false. The rules for propositional operators are exactly as in propositional logic. Quantifiers can be seen as a way of generalizing conjunction and disjunction. First-order variables range over some domain D. In the special case of finite domain $D = \{d_1, \ldots, d_k\}$, we have that

$$\begin{array}{rcl}(\forall x.P(x)) & \leftrightarrow & P(d_1) \land \ldots \land P(d_k) \\ (\exists x.P(x)) & \leftrightarrow & P(d_1) \lor \ldots \lor P(d_2)\end{array}$$

which corresponds to the intuitive definition of terms for all and there exists. Note, however, that D can be infinite in general (for example: the set of integers or reals).

Atomic formulas evaluate to true or false. We build atomic formulas by applying predicate symbols and equality to terms:

1. if t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula;
2. if t_1, , t_n are terms and P is a predicate symbol that takes n arguments, then $P(t_1, \ldots, t_n)$ is an atomic formula.

Predicates represent relations, for example, we can represent $\leq$ as a binary relation.

Terms denote elements of the domain D. We build them starting from variables and constants and applying function symbols:

1. each first-order variable is a term
2. a constant is a term
3. if $t_1, \ldots, t_n$ are terms and f is a function symbol that takes n arguments, then $f(t_1, \ldots, t_n)$ is also a term.

Example of constants are numerals for natural numbers, such as $0, 1, 2, \ldots$. Examples of function symbols are operations such as $+, -, /$.

From above we see that the set of formulas depends on the set of predicate and function symbols. This set is is called vocabulary or language.

5.4 Bounded Quantifiers

In general all quantifiers range over some universal domain D. To restrict them to subsets of D, we can use bounded quantifiers:

1. $\exists x \in S.P(x)$ means $\exists x.(x \in S \land P(x))$
2. $\forall x \in S.P(x)$ means $\forall x.(x \in S \rightarrow P(x))$ (note implication instead of conjunction)

More generally, if ρ is some binary relation written in infix form, such as $<, \leq, >$ $, \geq$ we write

1. $\exists x \, \rho \, t.P(x)$ meaning $\exists x.(x \, \rho \, t \land P(x))$
2. $\forall x \, \rho \, t.P(x)$ means $\forall x.(x \, \rho \, t \rightarrow P(x))$ (note implication instead of conjunction)

5.5 Evaluating First-Order Logic Formulas

To evaluate first-order logic formulas we need to know:

1. the domain set D
2. values of first-order variables in the formula
3. interpretation of predicate symbols and function symbols (e.g. does predicate symbol R denote $\leq$ or $<$ or $>$, does binary function symbol f denote $+$ or $-$)

5.5.1 Examples

Consider formula F given by $\forall x.(P(x) \vee P(f(x))$. The formula has one predicate symbol P that takes one urgument (we call it unary) and one unary function symbol f. Assume that the domain D is the set of integers and consider two possible interpretations:

1. P denotes property of being an even integer and f denotes the successor function $f(x) = x + 1$. Is formula true under this interpretation?
2. P denotes property of being an even integer and f denotes the squaring function $f(x) = x^2$. Is formula true under this interpretation?

Now consider formula $\forall x.\forall y.((P(x) \wedge x = f(y)) \rightarrow P(f(y))$ in each of these two interpretations.

5.6 (Finite) Validity and Satisfiability of First-Order Logic Formulas

Formula is valid if and only if it evaluates to true for all domains, values of its variables and interpretations of predicate and function symbols.

Formula is satisfiable if and only if it evaluates to true for some domains, values of its variables, interpretation of predicate and function symbols.

Formula is unsatisfiable if and only if it is false for all domains, values of its variables and interpretations of predicate and function symbols.

Note

1. a formula F is unsatisfiable if and only if it is not satisfiable
2. a formula F is valid if and only if $\neg F$ is unsatisfiable

Answers to some important algorithmic questions (not immediate):

1. There is no algorithm that given a first-order logic formula outputs yes when the formula is valid and no otherwise - validity of first-order logic formulas is undecidable
2. There exists an enumeration procedure that systematically lists all valid formulas (and only valid formulas) - validity of first-order logic formulas is enumerable

If instead of considering all domains we only consider finite domains (no natural numbers but only e.g. prefixes of natural numbers of the form $\{1, \ldots, n\}$) then we obtain notions of finite validity, finite satisfiability and finite unsatisfiability.

Note that because finite domains are a special case of possible domains, we have the following:

1. if a formula is finitely satisfiable, then it is satisfiable
2. if a formula is valid, then it is also finitely valid

Answers to some important algorithmic questions about finite satisfiability:

1. There is no algorithm that given a first-order logic formula outputs yes when the formula is satisfiable and no otherwise - finite satisfiability of first-order logic formulas is undecidable
2. There exists an enumeration procedure that systematically lists all finitely satisfiable formulas (and only finitely satisfiable formulas) - finite satisfiability of first-order logic formulas is enumerable

5.7 Some Valid First-Order Logic Formulas

If we take a propositional tautology and replace equal propositional variables with equal atomic formulas, we obtain a valid formula. Such formula is called an instance of a tautology.

But there are many other valid formulas. Such formulas are valid laws of thinking that we often use in mathematics (explicitly, or implicitly without mentioning them).

$$\begin{array}{l}
(\forall x.(P(x) \land Q(x)) \leftrightarrow ((\forall x.P(x)) \land (\forall x.Q(x))) \\
(\exists x.(P(x) \land Q(x)) \rightarrow ((\exists x.P(x)) \land (\exists x.Q(x))) \\
(\exists x.(P(x) \lor Q(x)) \leftrightarrow ((\exists x.P(x)) \lor (\exists x.Q(x))) \\
((\forall x.P(x)) \lor (\forall x.Q(x))) \rightarrow (\forall x.(P(x) \lor Q(x)) \\
(\exists x.\forall y.R(x,y)) \rightarrow (\forall y.\exists x.R(x,y)) \\
(\neg(\exists x.P(x))) \leftrightarrow (\forall x.(\neg P(x)) \\
(\neg(\forall x.P(x))) \leftrightarrow (\exists x.(\neg P(x)) \\
(\neg(\exists x \rho t.P(x))) \leftrightarrow (\forall x \rho t.(\neg P(x)) \\
(\neg(\forall x \rho t.P(x))) \leftrightarrow (\exists x \rho t.(\neg P(x)) \\
(\forall x.(x = t \rightarrow F(x))) \leftrightarrow F(t) \\
(\exists x.(x = t \land F(x))) \leftrightarrow F(t)
\end{array}$$

6 First-Order Logic Syntax

First order language $\mathcal{L}$ is a set of relation symbols R and function symbols f, each of which comes with arity $ar(R)$, $ar(f)$, which are ≥ 0. Function symbols of arity 0 are constants. Relation symbols of arity 0 are propositional variables.

The set V denotes countably infinite set of first-order variables, which is independent of the language.

$$\begin{array}{rcl}
F & ::= & A \mid \neg F \mid (F \land F) \mid (F \lor F) \mid (F \rightarrow F) \mid (F \leftrightarrow F) \\
 & \mid & \forall x.F \mid \exists x.F \\
A & ::= & R(T, \ldots, T) \mid (T = T) \\
T & ::= & V \mid f(T, \ldots, T)
\end{array}$$

Terminology summary:

1. F - first-order logic formula (in language $\mathcal{L}$)
2. A - atomic formula

3. $A, \neg A$ - literals
4. T - term
5. f - function symbol
6. R - relation symbol
7. V - first-order variables

Omitting parentheses:

1. $\wedge$, $\vee$ are associative
2. priorities, from strongest-binding: $(\neg)$; $(\wedge, \vee)$; $(\rightarrow, \leftrightarrow)$; $(\forall, \exists)$

When in doubt, use parentheses.
Example: Consider language $\mathcal{L} = \{P, Q, R, f\}$ with $ar(P) = 1$, $ar(Q) = 1$, $ar(R) = 2$, $ar(f) = 2$. Then

$$\neg \forall x.\, \forall y. R(x, y) \wedge Q(x) \rightarrow Q(f(y, x)) \vee P(x)$$

denotes

$$\neg(\forall x.\, (\forall y.((R(x, y) \wedge Q(x)) \rightarrow (Q(f(y, x)) \vee P(x)))))$$

Often we use infix notation for relation and function symbols. In the example above, if we write R and f in infix notation, the formula becomes

$$\neg \forall x.\, \forall y. x\, R\, y \wedge Q(x) \rightarrow Q(y\, f\, x) \vee P(x)$$

Notation: when we write $F_1 \equiv F_2$ this means that F_1 and F_2 are identical formulas (with identical syntax trees). For example, $p \wedge q \equiv p \wedge q$, but it is not the case that $p \wedge q \equiv q \wedge p$.
In Isabelle theorem prover we use this
ASCII notation for First-Order Logic
Usually we treat formulas as syntax trees and not strings.
FV denotes the set of free variables in the given propositional formula and can be defined recursively as follows:

$$\begin{array}{rcl}
FV(x) & = & \{x\},\ \text{for } x \in V \\
FV(f(t_1, \ldots, t_n)) & = & F(t_1) \cup \ldots \cup F(t_n) \\
FV(R(t_1, \ldots, t_n)) & = & F(t_1) \cup \ldots \cup F(t_n) \\
FV(t_1 = t_2) & = & F(t_1) \cup F(t_2) \\
FV(\neg F) & = & FV(F) \\
FV(F_1 \wedge F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(F_1 \vee F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(F_1 \rightarrow F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(F_1 \leftrightarrow F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(\forall x.F) & = & FV(F) \setminus \{x\} \\
FV(\exists x.F) & = & FV(F) \setminus \{x\}
\end{array}$$

If $FV(F) = \emptyset$, we call F a closed first-order logic formula, or sentence.

7 First-Order Logic Semantics

An interpretation for first-order logic language $\mathcal{L}$ is the pair $I = (D, \alpha)$ where D is a nonempty set, called the domain of interpretation, and α is the interpretation function, which assigns

1. to each first-order variable $x \in V$, an element $\alpha(x) \in D$
2. to each relation symbol $R \in \mathcal{L}$ with arity $ar(R) = n$, a relation $\alpha(R) \subseteq D^n$
3. to each function symbol $f \in \mathcal{L}$ with arity $ar(f) = n$, a function $\alpha(f) : D^n \to D$

If $I = (D, \alpha)$ we denote D by D_I and α by α_I.

Because terms denote values from domain D_I and formulas denote truth values from $\mathcal{B} = \{\mathit{false}, \mathit{true}\}$, we define two semantic evaluation functions:

1. $e_F : \mathcal{F} \to I \to \mathcal{B}$
2. $e_T : \mathcal{T} \to I \to D_I$

We evaluate terms by recursion on the structure of T:

$$\begin{array}{rcl} e_T(x)(I) & = & \alpha_I(x) \\ e_T(f(t_1, \ldots, t_n))(I) & = & \alpha_I(f)(e_T(t_1)(I), \ldots, e_T(t_2)(I)) \end{array}$$

We evaluate formulas by recursion on the structure of F:

$$\begin{array}{rcl} e_F(R(t_1, \ldots, t_n)(I) & = & (e_T(t_1)(I), \ \ldots, \ e_T(t_n)(I)) \ \in \ \alpha_I(R) \\ e_F(t_1 = t_2)(I) & = & (e_T(t_1)(I) = e_T(t_2)(I)) \\ e_F(F_1 \wedge F_2)(I) & = & e_F(F_1)(I) \wedge e_F(F_2)(I) \\ e_F(F_1 \vee F_2)(I) & = & e_F(F_1)(I) \vee e_F(F_2)(I) \\ e_F(\neg F)(I) & = & \neg e_F(F)(I) \end{array}$$

How do we evaluate quantifiers?

We generalize this notion as follows: if I is an interpretation and T is a set of first-order formulas, we write $e_S(T)(I) = \mathit{true}$ iff for every $F \in T$ we have $e_F(F)(I) = \mathit{true}$ (set is treated as infinite conjunction). This is a generalization because $e_S(\{F\})(I) = e_F(F)(I)$.

A terminological note: in algebra, an interpretation is often called a structure. Instead of using α mapping language $\mathcal{L} = \{f_1, \ldots, f_n, R_1, \ldots, R_m\}$ to interpretation of its symbols, the structure is denoted by a tuple $(D, \alpha(f_1), \ldots, \alpha(f_n), \alpha(R_1), \ldots, \alpha(R_n))$. For example, an interpretation with domain $\mathcal{N}$, with one binary operation whose interpretation is $+$ and one binary relation whose interpretation is $\leq$ can be written as $(\mathcal{N}, +, \leq)$. This way we avoid writing α all the time, but it becomes more cumbersome to describe correspondence between structures.

7.1 Examples

7.1.1 Example with Finite Domain

Consider language $\mathcal{L} = \{s, <\}$ where s is a unary function symbol ($ar(s) = 1$) and $<$ is a binary relation symbol ($ar(<) = 2$). Let $I = (D, \alpha)$ be given by

$$\begin{array}{rcl} D & = & \{0, 1, 2\} \\ \alpha(x) & = & 1 \\ \alpha(s) & = & \{(0, 1), (1, 2), (2, 0)\} \\ \alpha(<) & = & \{(0, 1), (0, 2), (1, 2)\} \end{array}$$

Let us evaluate the truth value of these formulas:

$$x < s(x)$$

$$\exists x.\neg(x < s(x))$$

$$\forall x.\exists y.x < y$$

7.1.2 Example with Infinite Domain

Consider language $\mathcal{L} = \{s, dvd\}$ where s is a unary function symbol ($ar(s) = 1$) and dvd is a binary relation symbol ($ar(dvd) = 2$). Let $I = (D, \alpha)$ where $D = \{7, 8, 9, 10, \ldots\}$. Let dvd be defined as the strictly divides relation:

$$\alpha(dvd) = \{(i, j).\ \exists k \in \{2, 3, 4, \ldots\}.\ j = k \cdot i\}$$

What is the truth value of this formula

$$\forall x.\ \exists y.\ dvd(x, y)$$

What is the truth value of this formula

$$\exists x.\ \forall y.dvd(x, y)$$

7.1.3 Domain Non-Emptiness

Let $I = (D, \alpha)$ be an arbitrary interpretation. Consider formula

$$(\forall x.P(x)) \rightarrow (\exists y.P(y))$$

What is its truth value in I? Which condition on definition of I did we use?

This formula is true with the assumption that D is not empty.

With an empty domain, this formula would be false. There are other problems, for instance how to evaluate a variable?.

7.2 Satisfiability, Validity, and Semantic Consequence

Definition (satisfiability of set): If T is a set of formulas, a model of T is an interpretation such that $e_S(T)$. A set T of first-order formulas is satisfiable if there exists a model for T. A set T is unsatisfiable (contradictory) iff it is not satisfiable (it has no model).

Note: taking $T = \{F\}$ we obtain notion of satisfiability for formulas.

Definition (semantic consequence): We say that a set of formulas T_2 is a semantic consequence of a set of formulas T_1 and write $T_1 \models T_2$, iff every model of T_1 is also a model of T_2.

Definition: Formula is valid, denoted $\models F$ iff $\emptyset \models F$.

Lemma: $\models F$ iff for every interpretation I we have $e_F(F)(I)$.

Lemma: A set T of formulas is unsatisfiable iff $T \models \mathit{false}$.

Lemma: Let T be a set of formulas and G a formula. Then $T \models \{G\}$ iff the set $T \cup \{\neg G\}$ is contradictory.

One of the central questions is the study of whether a set of formulas is contradictory, many basic questions reduce to this problem.

7.3 Consequence Set

Definition: The set of all consequences of T:

$$\mathit{Conseq}(T) = \{F \mid T \models F\}$$

Note $T \models F$ is equivalent to $F \in \mathit{Conseq}(T)$.

Lemma: The following properties hold:

$$\begin{array}{rcl}
T & \subseteq & \mathit{Conseq}(T) \\
T_1 \subseteq T_2 & \rightarrow & \mathit{Conseq}(T_1) \subseteq \mathit{Conseq}(T_2) \\
\mathit{Conseq}(\mathit{Conseq}(T)) & = & \mathit{Conseq}(T) \\
T_1 \subseteq \mathit{Conseq}(T_2) \wedge T_2 \subseteq \mathit{Conseq}(T_1) & \rightarrow & \mathit{Conseq}(T_1) = \mathit{Conseq}(T_2)
\end{array}$$

8 Normal Forms for First-Order Logic

Example Formula
We will look at the language $\mathcal{L} = \{P, R, a, f\}$ where

1. P is relation symbol of arity one
2. R is relation symbol of arity two
3. a is a constant
4. f is a function symbol of two arguments

Consider this formula in $\mathcal{L}$:

$$\begin{array}{l}(\forall x.\exists y.\ R(x,y)) \wedge \\ (\forall x.\forall y.\ R(x,y) \rightarrow \forall z.\ R(x,f(y,z))) \wedge \\ (\forall x.\ P(x) \vee P(f(x,a))) \\ \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\end{array}$$

We are interested in checking the validity of this formula (is it true in all interpretations). We will check the satisfiability of the negation of this formula (does it have a model):

$$\neg\Big(((\forall x.\exists y.\ R(x,y)) \wedge \\ (\forall x.\forall y.\ R(x,y) \rightarrow \forall z.\ R(x,f(y,z))) \wedge \\ (\forall x.\ P(x) \vee P(f(x,a)))\ ) \\ \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\Big)$$

We will first consider a range of techniques that allow us to convert such formula to simpler normal forms.

8.1 Negation Normal Form

In negation normal form of formula the negation applies only to atomic formulas.

Every FOL formula can be transformed in NNF using the formulas used for the same purpose in PL extended by two new ones :

1. $\neg\neg F \Leftrightarrow F$
2. $\neg\bot \Leftrightarrow \top$
3. $\neg\top \Leftrightarrow \bot$
4. $\neg(F_1 \wedge F_2) \Leftrightarrow \neg F1 \vee \neg F2$
5. $\neg(F_1 \vee F_2) \Leftrightarrow \neg F1 \wedge \neg F2$
6. $F1 \rightarrow F2 \Leftrightarrow \neg F1 \vee F2$
7. $F1 \leftrightarrow F2 \Leftrightarrow (F1 \rightarrow F2) \wedge (F2 \rightarrow F1)$
8. $\neg\forall x.F[x] \Leftrightarrow \exists x.\neg F[x]$
9. $\neg\exists x.F[x] \Leftrightarrow \forall x.\neg F[x]$

8.1.1 NNF of Example

$$\begin{array}{l}(\forall x.\exists y.\ R(x,y)) \wedge \\ (\exists x.\exists y.\ \neg R(x,y) \vee \forall z.\ R(x,f(y,z))) \wedge \\ (\forall x.\ P(x) \vee P(f(x,a))) \wedge \\ (\exists x.\forall y.\ \neg R(x,y) \vee \neg P(y))\end{array}$$

8.2 Prenex Normal Form

Prenex normal form has all quantifiers in front.

Prenex normal form (PNF) is a formula of the form

$$Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . G$$

where $Q_i \in \{\forall, \exists\}$ and G has no quantifiers.

Any FOL formula can be transformed to PNF. First convert it to NNF, then if several quantified variables or free variables have the same name rename them to fresh names, and finaly use the following formulas :

1. $(\forall x.F) \lor G \Leftrightarrow \forall x.(F \lor G)$
2. $(\forall x.F) \land G \Leftrightarrow \forall x.(F \land G)$
3. $(\exists x.F) \lor G \Leftrightarrow \exists x.(F \lor G)$
4. $(\exists x.F) \land G \Leftrightarrow \exists x.(F \land G)$

8.2.1 PNF of Example

$$\begin{array}{l}
(\forall x_1 . \exists y_1 .\ R(x_1, y_1)) \land \\
(\exists x_2 . \exists y_2 .\ \forall z .\ \neg R(x_2, y_2) \lor R(x_2, f(y_2, z))) \land \\
(\forall x_3 .\ P(x_3) \lor P(f(x_3, a))) \land \\
(\exists x_4 . \forall y_4 .\ \neg R(x_4, y_4) \lor \neg P(y_4))
\end{array}$$

8.3 Skolem Normal Form

Let $P : D \times D \to \{true, false\}$ be a predicate with two arguments.

Note that

$$\exists x. \forall y. P(y, x) \to \forall u. \exists v. P(u, v)$$

but converse implication does not hold (take as P relation $\leq$ or $>$ on natural numbers).

In general, we have this theorem:

$$\forall u. \exists v. P(u, v) \leftrightarrow \exists g. \forall u. P(u, g(u))$$

where $g : D \to D$ is a function.

Proof: ($\leftarrow$): For each u we take $f(u)$ as the witness v.

($\to$): We know there exists a witness v for each u. We define f to map u to one such witness v. (To prove that this is possible requires axiom of choice from set theory.)

Note also that satisfiability of formula F expresses existential quantification over function symbols and relation symbols.

Definition: Skolemization is the result of applying this transformation

$$\forall x_1, \dots, x_n . \exists y. F \ \rightsquigarrow \forall x_1, \dots, x_n . subst(\{y \mapsto g(x_1, \dots, x_n)\})(F)$$

to the entire PNF formula to eliminate all existential quantifiers. Above, g is a fresh function symbol. Denote $snf(F)$ the result of applying skolemization to formula F.

Lemma: A set of formulas S in prenex normal form is satisfiable iff the set $\{snf(F) \mid F \in S\}$ is satisfiable.

8.3.1 SNF for Example

$$\begin{array}{l}(\forall x.R(x,g(x))) \land \\ (\forall x.\forall y.\forall z.\ \neg R(x,y) \lor R(x,f(y,z))) \land \\ (\forall x.\ P(x) \lor P(f(x,a))) \land \\ (\forall y.\ \neg R(c,y) \lor \neg P(y))\end{array}$$

Note: it is better to do PNF and SNF for each conjunct independently.

8.4 CNF and Sets of Clauses

Let $snf(F)$ be $\forall x_1,\ldots,x_n.F$. Convert F to conjunctive normal form $C_1 \land \ldots \land C_m$. Then $snf(F)$ is equivalent to

$$(\forall x_1,\ldots,x_n.C_1) \land \ldots \land (\forall x_1,\ldots,x_n.C_m)$$

where each C_i is a disjunction of first-order literals. We call C_i (first-order) clause. For a given formula F, denote the set of such clauses in conjunctive normal form of $snf(pnf(F))$ by $clauses(F)$.

We omit universal quantifiers because all variables are universally quantified. We use a convention to denote variables by $x, y, z, \ldots$ and constants by $a, b, c, \ldots$.

Theorem: The set S is satisfiable iff the set

$$\bigcup_{F \in S} clauses(F)$$

is satisfiable.

8.4.1 Clauses for Example

1. $C_1 = R(x,g(x))$
2. $C_2 = \neg R(x,y) \lor R(x,f(y,z)))$
3. $C_3 = P(x) \lor P(f(x,a))$
4. $C_4 = \neg R(c,y) \lor \neg P(y)$

8.4.2 Another Example: Irreflexive Dense Linear Orders

Let $\mathcal{L} = \{less\}$ be binary relation (strictly less). We consider the following axioms for irreflexive partial order that is total and dense:

$$\begin{array}{rcl} IRef & \equiv & \forall x.\ \neg less(x,x) \\ Tra & \equiv & \forall x.\ \forall y.\ \forall z.\ less(x,y) \land less(y,z) \to less(x,z) \\ Total & \equiv & \forall x.\forall y.\ x \neq y \to less(x,y) \lor less(y,x) \\ Dense & \equiv & \forall x.\forall y.\ less(x,y) \to \exists z.\ less(x,z) \land less(z,y) \end{array}$$

Let us find clauses for these axioms :

$$\begin{array}{l}
\neg less(x_1, x_1) \\
\neg less(x_2, y_2) \lor \neg less(y_2, z_2) \lor less(x_2, z_2) \\
\neg(x_3 \neq y_3) \lor less(x_3, y_3) \lor less(y_3, x_3) \\
\neg less(x_4, y_4) \lor \ less(x_4, f(x_4, y_4)) \\
\neg less(x_4, y_4) \lor \ less(f(x_4, y_4), y_4)
\end{array}$$

9 Semantic Argument Method

Suppose we want to prove the validity of a propositional logic formula F. Several methods to do this exists, one of which is called the semantic argument method.

We start the proof by assuming that a falsifying interpretation exists:

$$I \not\models F$$

and try to show that this leads to a contradiction by applying semantic definitions of the logical connectives.

Thus, we obtain a set of proof rules:

$$\frac{I \models \neg F}{I \not\models F} \text{ and } \frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F \land G}{\begin{array}{l} I \models F \\ I \models G \end{array}} \text{ and } \frac{I \not\models F \land G}{I \not\models F \mid I \not\models G}$$

$$\frac{I \models F \lor G}{I \models F \mid I \models G} \text{ and } \frac{I \not\models F \lor G}{\begin{array}{l} I \not\models F \text{ and} \\ I \not\models G \end{array}}$$

$$\frac{I \models F \to G}{I \not\models F \mid I \models G} \text{ and } \frac{I \not\models F \to G}{I \models F \, I \not\models G}$$

$$\frac{I \models F \leftrightarrow G}{I \models F \land G \mid I \not\models F \lor G} \text{ and } \frac{I \not\models F \leftrightarrow G}{I \models F \land \neg G \mid I \models \neg F \land G}$$

$$\frac{\begin{array}{l} I \models F \\ I \not\models F \end{array}}{I \models \bot}$$

9.1 Extension to First-order logic

The proof rules above apply in addition to the following proof rules for the quantifiers:

$$\frac{I \models \forall x.F}{I \lhd \{x \mapsto v\} \models F}$$

for any v in the domain of the interpretation.

$$\frac{I \not\models \exists x.F}{I \lhd \{x \mapsto v\} \not\models F}$$

for any v in the domain of the interpretation.

$$\frac{I \models \exists x.F}{I \lhd \{x \mapsto v\} \models F}$$

for a fresh v in the domain of the interpretation.

$$\frac{I \not\models \forall x.F}{I \lhd \{x \mapsto v\} \not\models F}$$

for a fresh v in the domain of the interpretation.

10 Sets and Relations

10.1 Sets

Sets are unordered collection of elements.

We denote a finite set containing only elements a, b and c by $\{a, b, c\}$. The order and number of occurrences does not matter: $\{a, b, c\} = \{c, a, b\} = \{a, b, b, c\}$.

1. $a \in \{a, b, c\}$
2. $d \notin \{a, b, c\}$ iff $d \neq a \land d \neq b \land d \neq c$

Empty set: $\emptyset$. For every x we have $x \notin \emptyset$.

To denote large or infinite sets we can use set comprehensions: $\{x.\ P(x)\}$ is set of all objects with property P.

$$y \in \{x.P(x)\} \;\leftrightarrow\; P(y)$$

Notation for set comprehension: $\{f(x)|x.P(x)\} = \{y.(\exists x.y = f(x) \land P(x))\}$

Sometimes the binder x can be inferred from context so we write simply $\{f(x)|P(x)\}$. In general there is ambiguity in which variables are bound. (Example: what does the a in $f(a, b)$ refer to in the expression:

$$\{a\} \cup \{f(a, b) \mid P(a, b)\}$$

does it refer to the outerone a as in $\{a\}$ or is it a newly bound variable? The notation with dot and bar resolves this ambiguity.

Subset: $A \subseteq B$ means $\forall x.x \in A \to x \in B$

$$A \cup B = \{x.x \in A \lor x \in B\}$$
$$A \cap B = \{x.x \in A \land x \in B\}$$
$$A \setminus B = \{x.x \in A \land x \notin B\}$$

Boolean algebra of subsets of some set U (we define $A^c = U \setminus A$):

1. $\cup, \cap$ are associative, commutative, idempotent
2. neutral and zero elements: $A \cup \emptyset = A$, $A \cap \emptyset = \emptyset$
3. absorption: $A \cup A = A$, $A \cap A = A$
4. deMorgan laws: $(A \cup B)^c = A^c \cap B^c$, $(A \cap B)^c = A^c \cup B^c$
5. complement as partition of universal set: $A \cap A^c = \emptyset$, $A \cup A^c = U$
6. double complement: $(A^c)^c = A$

Which axioms are sufficient?

10.2 Infinte Unions and Intersections

Note that sets can be nested. Consider, for example, the following set S

$$S = \{\{p, \{q, r\}\}, r\}$$

This set has two elements. The first element is another set. We have $\{p, \{q, r\}\} \in S$. Note that it is not the case that

Suppose that we have a set B that contains other sets. We define union of the sets contained in B as follows:

$$\bigcup B = \{x.\ \exists a.a \in B \land x \in a\}$$

As a special case, we have

$$\bigcup \{a_1, a_2, a_3\} = a_1 \cup a_2 \cup a_3$$

Often the elements of the set B are computed by a set comprehension of the form $B = \{f(i).\ i \in J\}$. We then write

$$\bigcup_{i \in J} f(i)$$

and the meaning is

$$\bigcup \{f(i).\ i \in J\}$$

Therefore, $x \in \bigcup \{f(i).\ i \in J\}$ is equivalent to $\exists i.\ i \in J \land x \in f(i)$.

We analogously define intersection of elements in the set:

$$\bigcap B = \{x . \forall a.a \in B \to x \in a\}$$

As a special case, we have

$$\bigcap \{a_1, a_2, a_3\} = a_1 \cap a_2 \cap a_3$$

We similarly define intersection of an infinite family

$$\bigcap_{i \in J} f(i)$$

and the meaning is

$$\bigcap \{f(i).\, i \in J\}$$

Therefore, $x \in \bigcap\{f(i).\, i \in J\}$ is equivalent to $\forall i.\, i \in J \to x \in f(i)$.

10.3 Relations

Pairs:

$$(a, b) = (u, v) \iff (a = u \land b = v)$$

Cartesian product:

$$A \times B = \{(x, y) \mid x \in A \land y \in B\}$$

Relations r is simply a subset of $A \times B$, that is $r \subseteq A \times B$.
Note:

$$A \times (B \cap C) = (A \times B) \cap (A \times C)$$

$$A \times (B \cup C) = (A \times B) \cup (A \times C)$$

10.3.1 Diagonal relation

$\Delta_A \subseteq A \times A$, is given by

$$\Delta_A = \{(x, x) \mid x \in A\}$$

10.4 Set operations

Relations are sets of pairs, so operations $\cap, \cup, \setminus$ apply.

10.5 Relation Inverse

$$r^{-1} = \{(y, x) \mid (x, y) \in r\}$$

10.6 Relation Composition

$$r_1 \circ r_2 = \{(x, z) \mid \exists y.(x, y) \in r_1 \land (y, z) \in r_2\}$$

Note: relations on a set A together with relation composition and Δ_A form a monoid structure:

$$\begin{array}{l} r_1 \circ (r_2 \circ r_3) = (r_1 \circ r_2) \circ r_3 \\ r \circ \Delta_A = r = \Delta_A \circ r \end{array}$$

Moreover,

$$\emptyset \circ r = \emptyset = r \circ \emptyset$$

$$r_1 \subseteq r_2 \to r_1 \circ s \subseteq r_2 \circ s$$

$$r_1 \subseteq r_2 \to s \circ r_1 \subseteq s \circ r_2$$

10.7 Relation Image

When $S \subseteq A$ and $r \subseteq A \times A$ we define image of a set S under relation A as

$$S \bullet r = \{y.\ \exists x.x \in S \land (x, y) \in r\}$$

10.8 Transitive Closure

Iterated composition let $r \subseteq A \times A$.

$$\begin{aligned} r^0 &= \Delta_A \\ r^{n+1} &= r \circ r^n \end{aligned}$$

So, r^n is n-fold composition of relation with itself.

Transitive closure:

$$r^* = \bigcup_{n \geq 0} r^n$$

Equivalent statement: r^* is equal to the least relation s (with respect to $\subseteq$) that satisfies

$$\Delta_A \ \cup\ (s \circ r)\ \subseteq\ s$$

or, equivalently, the least relation s (with respect to $\subseteq$) that satisfies

$$\Delta_A \ \cup\ (r \circ s)\ \subseteq\ s$$

or, equivalently, the least relation s (with respect to $\subseteq$) that satisfies

$$\Delta_A \ \cup\ r \cup (s \circ s)\ \subseteq\ s$$

10.9 Some Laws in Algebra of Relations

$$(r_1 \circ r_2)^{-1} = r_2^{-1} \circ r_1^{-1}$$

$$r_1 \circ (r_2 \cup r_3) = (r_1 \circ r_2) \cup (r_1 \circ r_3)$$

$$(r^{-1})^* = (r^*)^{-1}$$

Binary relation $r \subseteq A \times A$ can be represented as a directed graph (A, r) with nodes A and edges r

Graphical representation of r^{-1}, r^*, and $(r \cup r^{-1})^*$ Equivalence relation r is relation with these properties:

1. reflexive: $\Delta_A \subseteq r$
2. symmetric: $r^{-1} \subseteq r$
3. transitive: $r \circ r \subseteq r$

Equivalence classes are defined by

$$x/r = \{y \mid (x, y) \in r$$

The set $\{x/r \mid x \in A\}$ is a partition:

1. each set non-empty
2. sets are disjoint
3. their union is A

Conversely: each collection of sets P that is a partition defines equivalence class by

$$r = \{(x, y) \mid \exists c \in P. x \in c \wedge y \in c\}$$

Congruence: equivalence that agrees with some set of operations.
Partial orders:

1. reflexive
2. antisymmetric: $r \cap r^{-1} \subseteq \Delta_A$
3. transitive

10.10 Functions

Example: an example function $f : A \to B$ for $A = \{a, b, c\}$, $B = \{1, 2, 3\}$ is

$$f = \{(a, 3), (b, 2), (c, 3)\}$$

Definition of function, injectivity, surjectivity.
$2^B = \{A \mid A \subseteq B\}$
$(A \to B) = B^A$ - the set of all functions from A to B. For $|B| > 2$ it is a strictly bigger set than B.
$(A \to B \to C) = (A \to (B \to C))$ (think of exponentiation on numbers)
Note that $A \to B \to C$ is isomorphic to $A \times B \to C$, they are two ways of representing functions with two arguments. $(C^B)^A = C^{B \times A}$
There is also isomorphism between

1. n-tuples $(x_1, \ldots, x_n) \in A^n$ and
2. functions $f : \{1, \ldots, n\} \to A$, where $f = \{(1, x_1), \ldots, (n, x_n)\}$

10.10.1 Function update

Function update operator takes a function $f : A \to B$ and two values $a_0 \in A$, $b_0 \in B$ and creates a new function $f[a_0 \mapsto b_0]$ that behaves like f in all points except at a_0, where it has value b_0. Formally,

$$f[a_0 \mapsto b_0](x) = \left\{ \begin{array}{l} b_0, \text{ if } x = a_0 \\ f(x), \text{ if } x \neq a_0 \end{array} \right\}$$

10.10.2 Domain and Range of Relations and Functions

For relation $r \subseteq A \times B$ we define domain and range of r:

$$dom(r) = \{x.\ \exists y.(x, y) \in r\}$$

$$ran(r) = \{y.\ \exists x.(x, y) \in r\}$$

Clearly, $dom(r) \subseteq A$ and $ran(r) \subseteq B$.

10.10.3 Partial Function

Notation: $\exists^{\leq 1} x.P(x)$ means $\forall x.\forall y.(P(x) \wedge P(y)) \rightarrow x = y$.

Partial function $f : A \hookrightarrow B$ is relation $f \subseteq A \times B$ such that

$$\forall x \in A.\exists^{\leq 1} y.\ (x, y) \in f$$

Generalization of function update is override of partial functions, $f \oplus g$

10.10.4 Range, Image, and Composition

The following properties follow from the definitions:

$$(S \bullet r_1) \bullet r_2 = S \bullet (r_1 \circ r_2)$$

$$S \bullet r = ran(\Delta_S \circ r)$$