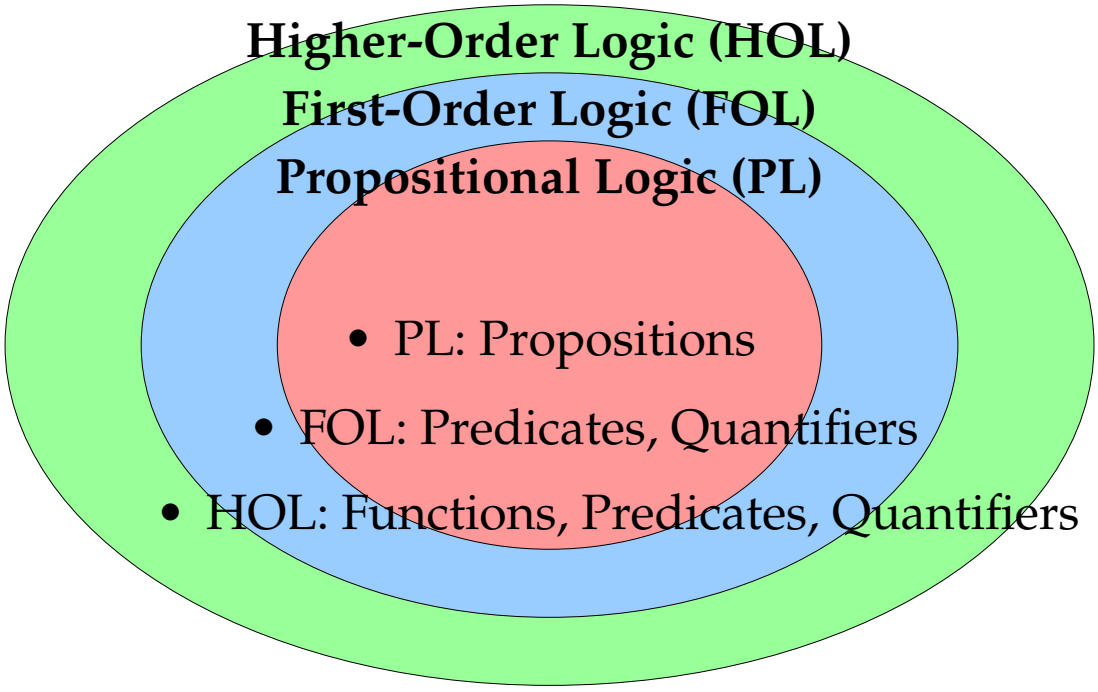
Software Verification

- Logic-based Program Verification -

Ji, Yong-Hyeon



A document presented for
the Software Verification

Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

July 6, 2024

Contents

1 Introduction **5**

2 Propositional Logic I **6**
2.1 Syntax and Semantic of Propositional Logic 6
2.1.1 Syntax 6
2.1.2 Semantics 7
2.1.3 Inductive Definition of Semantics 8
2.2 Satisfiability and Validity 8
2.2.1 Deciding Validity and Satisfiability 9
2.2.2 Deduction Rules for Propositional Logic 10
2.2.3 Proof Tree 11
2.2.4 Derived Rules 11

3 Propositional Logic II **12**
3.1 Equivalence and Implication, and Equisatisfiability 12
3.1.1 Equivalence and Implication 12
3.2 Substitution 13
3.2.1 Substitution 13
3.2.2 Semantic Consequences of Substitution 14
3.2.3 Composition of Substitution 14
3.3 Normal Forms: NNF, DNF, CNF 15
3.3.1 Negation Normal Form (NNF) 15
3.3.2 Disjunctive Normal Form (DNF) 15
3.3.3 Conjunctive Normal Form (CNF) 16
3.4 Decision Procedures for Satisfiability 16
3.4.1 Decision Procedures 16
3.4.2 Exhaustive Search 16
3.4.3 DPLL 17
3.4.4 Equisatisfiability 17
3.4.5 Conversion to an Equisatisfiable Formula in CNF 17

4 Problem Solving using SMT Solver **18**

5 First-Order Logic **19**
5.1 Introduction to FOL 19
5.1.1 Terms (Variables, Constants, and Functions) 19
5.1.2 Predicates 20
5.2 Syntax and Semantics of FOL 20
5.2.1 Syntax 20
5.2.2 Interpretation 21
5.2.3 Semantics of First-Order Logic 23

5.3 Satisfiability and Validity 23
5.4 Substitution 23
5.5 Normal Forms 23
5.6 Soundness, Completeness, and Decidability 23
5.7 First-Order Theories 23

A Boolean Functions **24**
A.1 Unary and Binary Boolean Function 24
A.2 Backus-Naur Form (BNF) 29

Symbols

In this paper, symbols are defined as follows.

$I \models P$ I satisfies P
$I \not\models P$ I does not satisfies P

Chapter 1

Introduction

[1] [2] [9]

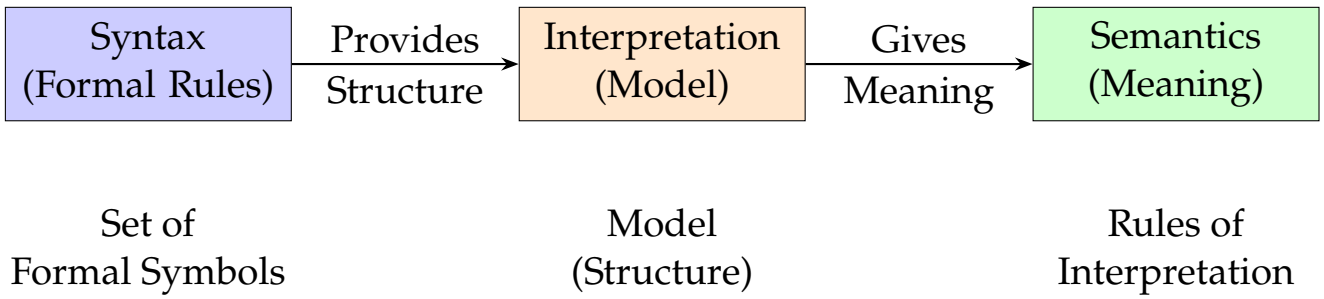
Chapter 2

Propositional Logic I

References:[3]

Propositional logic is a branch of logic that deals with propositions which can be either true or false. The basic components and operations in propositional logic are as follows:

2.1 Syntax and Semantic of Propositional Logic



2.1.1 Syntax

- **Atom**: basic elements
  - truth symbols $\bot$("false") and $\top$("true")
  - propositional variables $P, Q, R, \ldots$
- **Literal**: an atom α or its negation $\neg\alpha$.
- **Formula**: a literal or the application of a logical connective (boolean connectives) to formulas

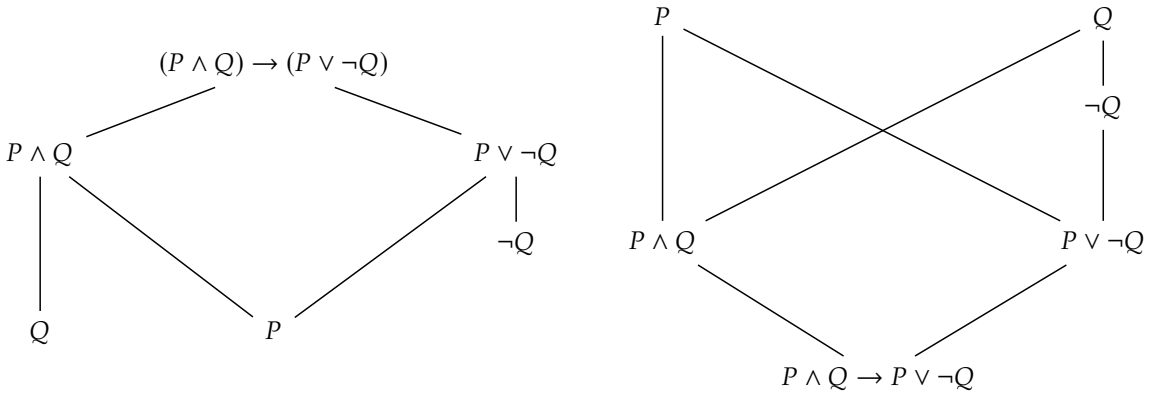
$$
\begin{array}{rcll}
F & \to & \bot & \\
 & | & \top & \\
 & | & P & \\
 & | & \neg F & \text{negation (“not”)} \\
 & | & F_1 \wedge F_2 & \text{conjunction (“and”)} \\
 & | & F_1 \vee F_2 & \text{disjunction (“or”)} \\
 & | & F_1 \to F_2 & \text{implication (“implies”)} \\
 & | & F_1 \leftrightarrow F_2 & \text{iff (“if and only if”)}
\end{array}
$$

- Formula G is a **subformula** of formula F if it occurs syntactically within G.

$$\begin{aligned}
\mathsf{sub}(\bot) &= \{\bot\} \\
\mathsf{sub}(\top) &= \{\top\} \\
\mathsf{sub}(P) &= \{P\} \\
\mathsf{sub}(\neg F) &= \{\neg F\} \cup \mathsf{sub}(F) \\
\mathsf{sub}(F_1 \wedge F_2) &= \{F_1 \wedge F_2\} \cup \mathsf{sub}(F_1) \cup \mathsf{sub}(F_2) \qquad \vdots
\end{aligned}$$

- Consider $F : (P \wedge Q) \rightarrow (P \vee \neg Q)$. Then

$$\mathsf{sub}(F) = \{F, P \wedge Q, P \vee \neg Q, P, Q, \neg Q\}.$$



- The strict subformulas of a formula are all its subformulas except itself.

- To minimally use parentheses, we define the relative precedence of the logical connectives from highest to lowest as follows:

$$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$$

- (Currying) Additionally, $\rightarrow$ and $\leftrightarrow$ associate to the right, e.g.,

$$P \rightarrow Q \rightarrow R \iff P \rightarrow (Q \rightarrow R).$$

- Example:
  - $(P \wedge Q) \rightarrow (P \vee \neg Q) \iff P \wedge Q \rightarrow P \vee \neg Q$
  - $(P_1 \wedge ((\neg P_2) \wedge \top)) \vee ((\neg P_1) \wedge P_2) \iff P_1 \wedge P_2 \top \vee \neg P_1 \wedge P_2$

2.1.2 Semantics

- The semantics of a logic provides its meaning. The meaning of a PL formula is either true or false.
- The semantics of a formula is defined with an **interpretation** (or assignment) that assigns truth values to propositional variables.
- For example, $F : P \wedge Q \rightarrow P \vee \wedge Q$ evaluates to true under the interpretation $I : \{P \mapsto \text{true}, Q \mapsto \text{false}\}$:
- The tabular notation is unsuitable for predicate logic. Instead, we define the semantics inductively.

P	Q	$\neg Q$	$P \wedge Q$	$P \vee \neg Q$	F
1	0	1	0	1	1

2.1.3 Inductive Definition of Semantics

In an inductive definition, the meaning of basic elements is defined first. The meaning of complex elements is defined in terms of subcomponents.

- We write $I \models F$ if F evaluates to **true** under I.
- We write $I \not\models F$ if F evaluates to **false** under I.

Note that

$$
\begin{array}{ll}
I \models \top, \quad I \not\models \bot, & \\
I \models P & \text{iff } I[P] = \text{true} \\
I \models \neg F & \text{iff } I[P] = \text{false} \\
I \models \neg F & \text{iff } I \not\models F \\
I \models F_1 \wedge F_2 & \text{iff } I \models F_1 \text{ and } I \models F_2 \\
I \models F_1 \vee F_2 & \text{iff } I \models F_1 \text{ or } I \models F_2 \\
I \models F_1 \rightarrow F_2 & \text{iff } I \not\models F_1 \text{ or } I \models F_2 \\
I \models F_1 \leftrightarrow F_2 & \text{iff } (I \models F_1 \text{ and } I \models F_2) \text{ or } (I \not\models F_1 \text{ and } I \not\models F_2)
\end{array}
$$

Example 2.1.1. Consider the formula

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

and the interpretation

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}\}.$$

The truth value of F is computed as follows:

1.	$I \models P$	since $I[P] = \text{true}$
2.	$I \not\models Q$	since $I[P] = \text{true}$
3.	$I \models \neg Q$	by 2 and semantics of $\neg$
4.	$I \not\models P \wedge Q$	by 2 and semantics of $\wedge$
5.	$I \models P \vee \neg Q$	by 1 and semantics of $\vee$
6.	$I \models F$	by 4 and semantics of $\rightarrow$

2.2 Satisfiability and Validity

- A formula F is **satisfiable** iff there exists an interpretation I such that $I \models F$.
- A formula F is **valid** iff for all interpretations I, $I \models F$.
- Satisfiability and validity are dual:

$$F \text{ is valid} \iff \neg F \text{ is unsatisfiable}$$

Proof. content... □

- We can check satisfiability by deciding validity, and vice versa.

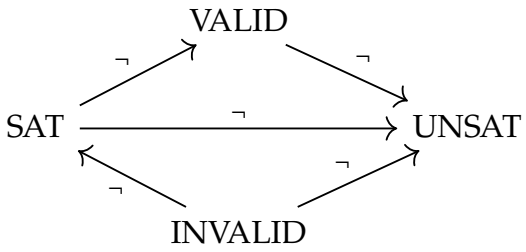
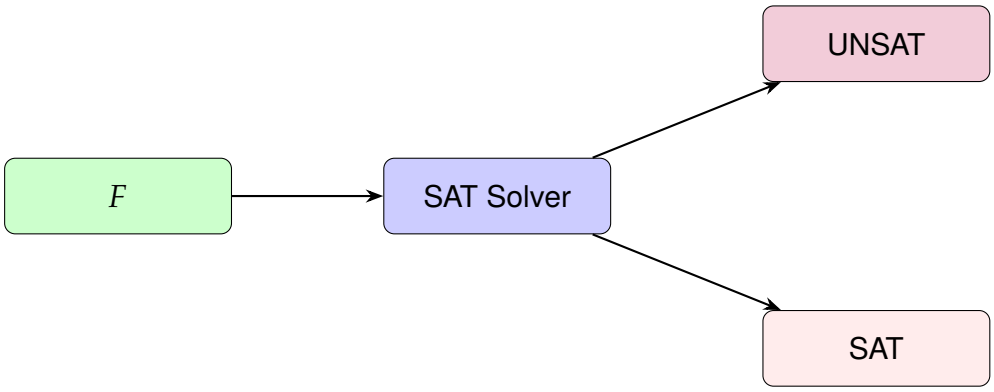


Figure 2.1: Relation of SAT, UNSAT, VALID and INVALID



2.2.1 Deciding Validity and Satisfiability

Two approaches to show F is valid:

- **Truth Table Method** performs exhaustive search: e.g.,

$$F : P \wedge Q \rightarrow P \vee \neg Q.$$

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

  Non-applicable to logic with infinite domain (e.g., first-order logic).

- **Semantic Argument Method** uses deduction:
  - Assume F is valid: $I \not\models F$ for some I (falsifying interpretation).
  - Apply deduction rules (proof rules) to derive a contradiction.
  - If every branch of the proof derives a contradiction, then F is valid.
  - If some branch of the proof never derives a contradiction, then F is invalid.

2.2.2 Deduction Rules for Propositional Logic

Negation Elimination

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

This rule shows how to derive F from $\neg F$. It eliminates the negation by asserting that if $\neg F$ holds, then F cannot hold.

Negation Introduction

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

This rule introduces F from the fact that $\neg F$ does not hold. It asserts that if the negation of F does not hold, then F must hold.

Conjunction Elimination[a]

$$\frac{I \models F \land G}{I \models F, I \models G}$$

This rule breaks down a conjunction into its individual components, showing that if $F \land G$ is true, then both F and G are true.

De Morgan's Law for Conjunction[a]

$$\frac{I \not\models F \land G}{I \not\models F \mid I \not\models G}$$

This rule states that if $F \land G$ is not true, then at least one of F or G is not true. This is an application of De Morgan's laws.

Disjunction Elimination[b]

$$\frac{I \models F \lor G}{I \models F \mid I \models G}$$

This rule asserts that if $F \lor G$ is true in an interpretation, then either F is true, or G is true, or both are true. This is also sometimes referred to as the rule of cases.

De Morgan's Law for Disjunction[b]

$$\frac{I \not\models F \lor G}{I \not\models F, I \not\models G}$$

This rule states that if $F \lor G$ is not true, then both F is not true and G is not true. This is directly derived from De Morgan's laws in classical logic.

Implication Elimination[c]

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

This rule states that if $F \to G$ is true in an interpretation, then either F is false or G is true. This corresponds to the definition of material implication in classical logic.

Denial of Implication[c]

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

This rule states that if $F \to G$ is not true in an interpretation, then F must be true and G must be false. This is the contrapositive form of material implication.

Biconditional Elimination[d]

$$\frac{I \models F \leftrightarrow G}{I \models F \land G \mid I \models \neg F \land \neg G}$$

This rule states that if $F \leftrightarrow G$ is true in an interpretation, then either both F and G are true, or both F and G are false. This corresponds to the definition of a biconditional statement in classical logic.

Negation of Biconditional[d]

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

This rule states that if $F \leftrightarrow G$ is not true in an interpretation, then F and G have opposite truth values, i.e., either F is true and G is false, or F is false and G is true. This aligns with the concept of exclusive or (XOR).

[a] And-Elimination
[b] Or-Elimination
[c] Material Implication
[d] If and Only If Elimination

[a] Contrapositive of Conjunction Introduction
[b] Contrapositive of Disjunction Introduction
[c] Contrapositive of Implication
[d] Exclusive Or Elimination

Contradiction Introduction (Proof by Contradiction)

$$\frac{I \models F \quad I \not\models F}{I \models \bot}$$

Example 2.2.1. To prove that the formula

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

is valid, assume that it is invalid and derive a contradiction:

1.	$I \not\models P\wedge \rightarrow P \vee \neg Q$	assumption
2.	$I \models P \wedge Q$	by 1 and semantics of $\rightarrow$
3.	$I \not\models P \vee \neg Q$	by 1 and semantics of $\rightarrow$
4.	$I \models P$	by 2 and semantics of $\wedge$
5.	$I\neg \models P$	by 3 and semantics of $\vee$
6.	$I \models \bot$	4 and 5 are contradictory

Example 2.2.2. To prove that the formula

$$F : (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$$

is valid, assume that it is invalid and derive a contradiction:

2.2.3 Proof Tree

A proof evolves as a tree.

- A branch is a sequence descending from the root.
- A branch is *closed* it contains a contradiction. Otherwise, the branch is *open*.
- It is a proof of the validity of F if every branch is closed; otherwise, each open branch describes a falsifying interpretation of F.

2.2.4 Derived Rules

The proof rules are sufficient, but **derived rules** can make proofs more concise. E.g., the rule of modus ponens:

$$\frac{I \models F \quad I \models F \rightarrow G}{I \models G}$$

The proof of the validity of the formula:

$$F : (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$$

1.	$I \not\models F$	assumption
2.	$I \models (P \rightarrow Q) \wedge (Q \rightarrow R)$	by 1 and semantics of $\rightarrow$
3.	$I \models P \rightarrow R$	by 1 and semantics of $\rightarrow$
4.	$I \models P$	by 3 and semantics of $\rightarrow$
5.	$I \not\models R$	by 3 and semantics of $\rightarrow$
6.	$I \not\models P \rightarrow Q$	2 and semantics of $\wedge$
7.	$I \not\models Q \rightarrow R$	2 and semantics of $\wedge$
8.	$I \not\models Q$	by 4, 6, and modus ponens
9.	$I \not\models R$	by 8, 7, and modus ponens
10.	$I \models \bot$	5 and 9 are contradictory

Chapter 3

Propositional Logic II

[3] [4] [5]

3.1 Equivalence and Implication, and Equisatisfiability

3.1.1 Equivalence and Implication

- Two formulas F_1 and F_2 are equivalent

$$F_1 \iff F_2$$

  iff $F_1 \leftrightarrow F_2$ is valid, i. e., for all interpretations I, $I \models F_1 \leftrightarrow F_2$.

- Formula F_1 implies formula F_2

$$F_1 \implies F_2$$

  iff $F_1 \rightarrow F_2$ is valid, i. e., for all interpretations I, $I \models F_1 \rightarrow F_2$.

- $F_1 \iff F_2$ and $F_1 \implies F_2$ are not formulas. They are semantic assertions.
- We can check equivalence and implication by checking satisfiability.

Example 3.1.1.

- $P \iff \neg\neg P$ means that $P \leftrightarrow \neg\neg p$ is valid.
- $P \rightarrow Q \iff \neg P \vee Q$ means that $P \rightarrow Q \leftrightarrow \neg P \vee Q$ is valid.

Exercise 3.1.1. Prove that

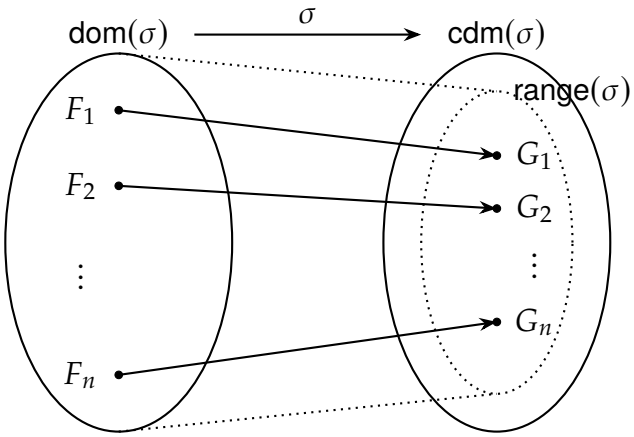
$$R \wedge (\neg R \vee P) \implies P.$$

Sol. We want to show that $R \wedge (\neg R \vee P) \rightarrow P$ is valid.

1. $I \models R \wedge (\neg R \vee P)$
2. $I \not\models P$
3. $I \models R$
4.

□

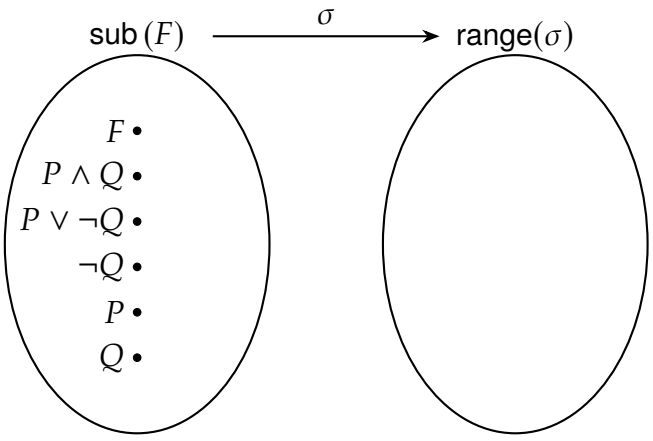
3.2 Substitution



3.2.1 Substitution

- A substitution σ is a mapping from formulas to formulas:
$$\sigma : \{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\}$$
- The domain of σ, $\mathsf{dom}(\sigma)$, is
$$\mathsf{dom}(\sigma) = \{F_1, \ldots, F_n\}$$
while the range is $\mathsf{range}(\sigma)$
$$\mathsf{range}(\sigma) : \{G_1, \ldots, G_n\}\,.$$
- The application of a substitution σ to a formula F, $F\sigma$, replaces each occurence of F_i with G_i. Replacements occur all at once.
- When two subformulas F_j and F_k are in $\mathsf{dom}(\sigma)$ and F_k strict subformula of F_j, then F_j is replaced first.

Example 3.2.1. Consider formula
$$F : P \wedge Q \rightarrow P \vee \neg Q$$
and substitution
$$\sigma : \{P \mapsto R, P \wedge Q \mapsto P \rightarrow Q\}\,.$$
Then
$$F\sigma : (P \rightarrow Q) \rightarrow R \vee \neg Q.$$
Note that $F\sigma \neq (R \rightarrow Q) \rightarrow R \vee \neg Q$.



- A variable substitution is a substitution in which the domain consists only of propositional variables.
- When we write $F[F_1, \ldots, F_n]$, we mean that formula F and have formulas $F_1, \ldots, F_n$ as subformulas.
- $$\sigma : \{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\} \implies F[F_1, \ldots, F_n]\sigma : F[G_1, \ldots, G_n].$$
- For example, in the previous example, writing

  $$F[P, P \wedge Q]\sigma : F[R, P \to Q]$$

  emphasizes that P and $P \wedge Q$ are replaced by R and $P \to Q$, respectively.

3.2.2 Semantic Consequences of Substitution

Substitution of Equivalent Formulas

Lemma 3.2.1 *Consider substitution $\sigma : \{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\}$ such that for each i, $F_i \iff G_i$. Then,*

$$F \iff F\sigma$$

Example 3.2.2. Applying $\sigma : \{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\}$ to $F : (P \to Q) \to R$ produces $(\neg P \vee Q) \to R$ that is equivalent to F.

Valid Template

Lemma 3.2.2 *If F is valid and $G = F\sigma$ for some variable substitution σ, then G is valid.*

Example 3.2.3. Because $F : (P \to Q) \leftrightarrow (\neg P \vee Q)$ is valid, every formula of the form $F_1 \to F_2$ is equivalent to $\neg F_1 \vee F_2$, for arbitrary formulas F_1 and F_2.

3.2.3 Composition of Substitution

Given substitutions σ_1 and σ_2, their composition $\sigma = \sigma_1\sigma_2$ ("apply σ_1 and then σ_2") is computed as follows:

1. Apply σ_2 to each formula of the range of σ_1, and add the results to σ.
2. If F_i of $F_i \mapsto G_i$ appears in the domain of σ_2 but not in the domain of σ_1, then add $F_i \mapsto G_i$ to σ.

Example 3.2.4.

$$\begin{aligned}\sigma_1\sigma_2 : &\{P \mapsto R, P \wedge Q \mapsto P \to Q\}\,\{P \mapsto S, S \mapsto Q\} \\ &= \left\{P \mapsto R\sigma_2, P \wedge Q \mapsto (P \to Q)\sigma_2, S \mapsto Q\right\} \\ &= \{P \mapsto R, P \wedge Q \mapsto S \to Q, S \mapsto Q\}\,.\end{aligned}$$

3.3 Normal Forms: NNF, DNF, CNF

A normal form of formulas is a syntactic restriction such that for every formula of the logic, there is an equivalent formula in the normal form. Three useful normal forms in logic:

- **Negation Normal Form (NNF)**
- **Disjunctive Normal Form (DNF)**
- **Conjunctive Normal Form (CNF)**

3.3.1 Negation Normal Form (NNF)

- NNF requires that $\neg, \wedge$, and $\vee$ are the only connective (i.e., no $\rightarrow$ and $\leftrightarrow$) and that negations are only applied to variables.

  (✓) $P \wedge Q \wedge (R \vee \neg S)$

  (✗) $\neg P \vee \neg(P \wedge Q)$

  (✗) $\neg\neg P \wedge Q$

- Transforming a formula F to equivalent formula F' in NFF can be done by repeatedly applying (left-to-right) the following template equivalences:

$$\begin{aligned}
\neg\neg F_1 &\iff F_1 \\
\neg\top &\iff \bot \\
\neg\bot &\iff \top \\
\neg(F_1 \wedge F_2) &\iff \neg F_1 \vee \neg F_2 \\
\neg(F_1 \vee F_2) &\iff \neg F_1 \wedge \neg F_2 \\
F_1 \rightarrow F_2 &\iff \neg F_1 \vee F_2 \\
F_1 \leftrightarrow F_2 &\iff (\neg F_1 \vee F_2) \wedge (F_1 \vee \neg F_2)
\end{aligned}$$

Exercise 3.3.1. Convert $F : \neg(P \rightarrow \neg(P \wedge Q))$ into NNF.

3.3.2 Disjunctive Normal Form (DNF)

- A formula is in disjunction normal form (DNF) if it is a disjunction of conjunctions of literals

$$\bigvee_i \bigwedge_j l_{i,j}.$$

- To convert a formula F into an equivalent formula in DNF, transform F into NNF and then distribute conjunctions over disjunctions:

$$\begin{aligned}
(F_1 \vee F_2) \wedge F_3 &\iff (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\
F_1 \wedge (F_2 \vee F_3) &\iff (F_1 \wedge F_2) \vee (F_1 \wedge F_3)
\end{aligned}$$

Exercise 3.3.2. To convert

$$F : (Q_1 \vee \neg\neg Q_2) \wedge (\neg R_1 \rightarrow R_2)$$

into DNF. [Hint] First transform it into NNF, then apply distributivity.

3.3.3 Conjunctive Normal Form (CNF)

- A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals:

$$\bigwedge_i \bigvee_j l_{i,j}$$

  where each disjunction of literals is called a **clause**.

- To convert a formula F into an equivalent formula in DNF, transform F into NNF and distribute disjunctions over conjunctions:

$$(F_1 \wedge F_2) \vee F_3 \iff (F_1 \vee F_3) \wedge (F_2 \vee F_3)$$
$$F_1 \vee (F_2 \wedge F_3) \iff (F_1 \vee F_2) \wedge (F_1 \vee F_3)$$

3.4 Decision Procedures for Satisfiability

3.4.1 Decision Procedures

- A **decision procedure** decides whether F is satisfiable after some finite steps of computation.

- Approaches for deciding satisfiability:

  - **Search**: exhaustively search through all possible assignments
  - **Deduction**: deduce facts from known facts by iteratively applying proof rules
  - **Combination**: Modern SAT solvers are based on DPLL that combines search and deduction in an effective way

3.4.2 Exhaustive Search

- The recursive algorithm for deciding satisfiability:

- When applying $F\{P \mapsto \top\}$ and $F\{P \mapsto \bot\}$, the resulting formulas should be simplified using template equivalence on PL:

Example 3.4.1. Consider

$$F : (P \to Q) \wedge \neg P.$$

1. Choose P and recurse on the first case:

$$F\{P \mapsto \top\} : (\top \to Q) \wedge \neg\top$$

   which is equivalent to $\bot$.

2. Try the other case:

$$F\{P \to \bot\} : (\bot \to Q) \wedge \neg\bot$$

   which is equivalent to $\top$.

3. Arbitrarily assigning a value to Q produces the satisfying interpretation.

3.4.3 DPLL

3.4.4 Equisatisfiability

- SAT solver convert a given formula F to CNF.
- Conversion to an equivalent CNF incurs exponential blow-up in worst-case.
- F is converted to and equisatisfiable CNF formula, which increase the size by only a constant factor.
- F and F' are **equisatisfiable** when F is satisfiable iff F' is satisfiable.
- Equisatisfiability is weaker notion of equivalence, which is still useful when deciding satisfiability.

3.4.5 Conversion to an Equisatisfiable Formula in CNF

Chapter 4

Problem Solving using SMT Solver

[6] [7] [8]

Chapter 5

First-Order Logic

5.1 Introduction to FOL

- An extension of propositional logic with predicates, functions, and quantifiers.
- First-order logic is also called predicate logic, first-order predicate calculus, and relational logic.
- First-order logic is expressive enough to reason about programs.
- However, completely automated reasoning is not possible.

5.1.1 Terms (Variables, Constants, and Functions)

- Terms denote the objects that we are reasoning about.
- While formulas in PL evaluate to true or false, term in FOL evaluate to values in an underlying domain such as integers, strings, lists, etc.
- Terms in FOL are defined by the grammar:

$$t \to x \mid c \mid f(t_1, \ldots, f_n).$$

  - Basic terms are **variables** $(x, y, z, \ldots)$ and **constant** $(a, b, c, \ldots)$.
  - Composite terms include n-ary **functions** applied to n terms, i. e., $f(t_1, \ldots, d_n)$, where t_1s are terms.
    - A constant can be viewed as a 0-ary (null-ary) function.

Example 5.1.1.

Symbol	Meaning
$f(a)$	A unary function f applied to a constant
$g(x, b)$	A binary function g applied to a variable x and a constant b
$f(g(x, f(b)))$	

5.1.2 Predicates

- The propositional variables of PL are generalized to **predicates** in FOL, denoted $p, q, r, \ldots$.
- An n-ary predicates take n terms as arguments.
- A FOL propositional variable is 0-ary predicate, denoted $P, Q, \ldots$.

Example 5.1.2.

Symbol	Meaning
P	A propositional variable (or 0-ary predicate)
$p(f(x), g(x, f(x)))$	A binary predicate applied to two terms

5.2 Syntax and Semantics of FOL

5.2.1 Syntax

- **Atom**: basic elements
  - truth symbols $\bot$("false") and $\top$("true")
  - n-ary predicates applied to n terms.
- **Literal**: an atom α or its negation $\neg\alpha$.
- **Formula**: a literal or the application of a logical connective to formulas or the application of a quantifier to a formula.

$$
\begin{array}{rcll}
F & \to & \bot \mid \top \mid p(t_1, \ldots, t_n) & \text{atom} \\
& \mid & \neg F & \text{negation ("not")} \\
& \mid & F_1 \wedge F_2 & \text{conjunction ("and")} \\
& \mid & F_1 \vee F_2 & \text{disjunction ("or")} \\
& \mid & F_1 \to F_2 & \text{implication ("implies")} \\
& \mid & F_1 \leftrightarrow F_2 & \text{iff ("if and only if")} \\
& \mid & \exists x.F[x] & \text{existential quantification} \\
& \mid & \forall x.F[x] & \text{universal quantification}
\end{array}
$$

Remark 5.2.1 (Notation on Quantification)**.**

- In $\forall x.F[x]$ and $\exists x.F[x]$, x is the **quantified variable** and $F[x]$ is the **scope** of the quantifier. We say x in $F[x]$ is **bound**.
- $\forall x.\forall y.F[x, y]$ is often abbreviated by $\forall x, y.F[x, y]$.
- The scope of the quantified variable extends as far as possible: e.g.,

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

- A variable is **free** in $F[x]$ if it is not bound. $\mathsf{free}(F)$ and $\mathsf{bound}(F)$ denote the free and bound variables of F, respectively. A formula F is **closed** if F has no free variables. E.g.,

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

- If $\mathsf{free}(F) = \{x_1, \ldots, x_n\}$, then its **universal closure** is $\forall x_1 \ldots \forall x_n.F$ and its **existential closure** is $\exists x_1 \ldots \exists x_n.F$ They are usually written $\forall * .F$ and $\exists * .F$.

Example 5.2.1 (FOL Formulas)**.**

- Every cat has its day.

$$\forall x.cat(x) \to \exists y.day(y) \wedge itsDay(x, y)$$

-
-
- Fermat's Last Theorem

5.2.2 Interpretation

Observation. Consider

$$\begin{aligned} x + y &> z, \\ y &> z - x, \end{aligned}$$

i. e.,

$$F : (x + y > z) \to (y > z - x).$$

- Note $+, -, >$ are just symbols: we could written

$$\begin{aligned} [+(x, y) > z] &\to [y > -(z, x)], \\ [> (+(x, y), z)] &\to [> (y, -(z, x))], \\ p(f(x, y), z) &\to p(y, g(z, x)). \end{aligned}$$

- Domain: $D_I = \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- Assignment:

$$\begin{aligned} \alpha_I = \{ & + \mapsto +_{\mathbb{Z}}, \\ & - \mapsto -_{\mathbb{Z}}, \\ & > \mapsto >_{\mathbb{Z}}, \\ & x \mapsto 13, \\ & y \mapsto 42, \\ & z \mapsto 1, \\ & \ldots \\ \} & \end{aligned}$$

Interpretation

Definition 5.2.1. A FOL **interpretation** $I : (D_I, \alpha_I)$ is a pair of a domain and an assignment.

- D_I is a nonempty set of values such as integers, real numbers, etc.
- α_I maps variables, constant, function, and predicate symbols to elements, functions, and predicates over D_I.
  - each variable x is assigned a value from D_I
  - each n-ary function symbol f assigned an n-ary function $f_I : D_I^n \to D_I$.
  - each n-ary predicate symbol p is assigned an n-ary predicate $p_I : D_I^n \to \{\text{true}, \text{false}\}$
- Arbitrary terms and atoms are evaluated recursively:

$$\alpha_I[f(t_1, \ldots, t_n)] = \alpha_I[f](\alpha_I[t_1], \ldots, \alpha_I[t_n]),$$
$$\alpha_I[p(t_1, \ldots, t_n)] = \alpha_I[p](\alpha_I[t_1], \ldots, \alpha_I[t_n]).$$

5.2.3 Semantics of First-Order Logic

Observation. Consider

$$F : (x + y > z) \rightarrow (y > z - x).$$

- Domain: $D_I = \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- Assignment:

$$\begin{aligned} \alpha_I = \{ & + \mapsto +_{\mathbb{Z}}, \\ & - \mapsto -_{\mathbb{Z}}, \\ & > \mapsto >_{\mathbb{Z}}, \\ & x \mapsto 13, \\ & y \mapsto 42, \\ & z \mapsto 1, \\ & \ldots \\ \} & \end{aligned}$$

1. $I \models x + y > z$ since $\alpha_I[x + y > z] = 13 +_{\mathbb{Z}} 42 >_{\mathbb{Z}} 1$
2. $I \models y > z - x$ since $\alpha_I[x + y > z] = 42 >_{\mathbb{Z}} 1 -_{\mathbb{Z}} 13$
3. $I \models F$ by 1,2, and the semantics of $\rightarrow$

Observation. Consider

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

and the interpretation $I : (D : \{v_1, v_2\}, \alpha_I)$:

$$\begin{aligned} \alpha_I = \{ & f(v_1) \mapsto v_1, \\ & f(v_2) \mapsto v_2, \\ & g(v_1) \mapsto v_2, \\ & g(v_2) \mapsto v_1\} \end{aligned}$$

Compute the truth value of F under I as follows:

1. $I \triangleright \{x \mapsto v\} \not\models f(x) = g(x)$ for all $v \in D$
2. $I \not\models \exists x.f(x) = g(x)$ by the semantics of $\exists$

5.3 Satisfiability and Validity

5.4 Substitution

5.5 Normal Forms

5.6 Soundness, Completeness, and Decidability

5.7 First-Order Theories

Appendix A

Boolean Functions

A.1 Unary and Binary Boolean Function

Propositional Variable

Definition A.1.1. A propositional variable is an element of the set $\{0, 1\}$.

Example A.1.1. $P, Q, R, \ldots, \in \{0, 1\}$ and so on.

Truth Function

Definition A.1.2. Let $\mathbb{B} = \{0, 1\}$ be the *boolean domain*. Let $k \in \mathbb{N}$. A mapping

$$f : \mathbb{B}^k \to \mathbb{B}$$

is called a **truth function**.

Count of Truth Functions

Proposition A.1.1 *There are $2^{(2^k)}$ distinct truth functions on $k \in \mathbb{N}$ variables.*

Proof. Let $f : \mathbb{B}^k \to \mathbb{B}$ be a truth function for $k \in \mathbb{N}$. Then

(Cardinality of Cartesian Product of Finite Sets)

$$\#(\mathbb{B}^k) = \#(\overbrace{\mathbb{B} \times \cdots \times \mathbb{B}}^{k \text{ times}}) = \overbrace{\#\mathbb{B}\#\mathbb{B} \cdots \#\mathbb{B}}^{k \text{ times}} = \overbrace{2 \cdot 2 \cdots 2}^{k \text{ times}} = 2^k.$$

(Cardinality of Set of All Mappings.)

$$\#(T^S) = \#\left\{f \subseteq S \times T : f \text{ is a mapping}\right\} = (\#T)^{(\#S)} \implies \#(\mathbb{B})^{\#(\mathbb{B}^k)} = 2^{(2^k)}$$

□

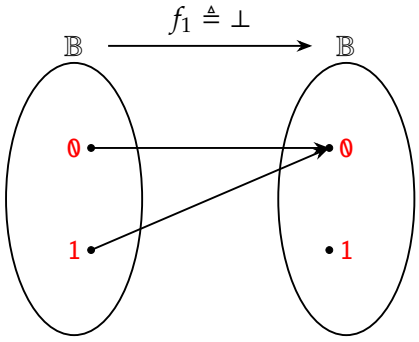
Unary Truth Functions

Corollary A.1.1 *There are 4 distinct unary truth functions:*

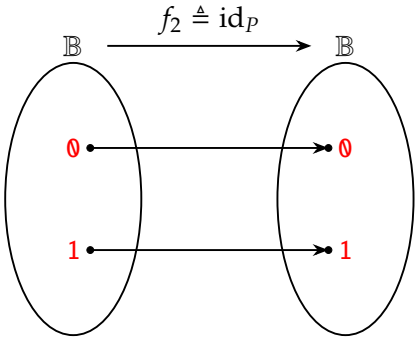
- *The contradiction function* $f(P) = \mathbb{0}$
- *The tautology function* $f(P) = 1$
- *The identity function* $f(P) = P$
- *The logical not function* $f(P) = \neg P$

Proof. By Count of Truth Functions, there are $2^{(2^1)} = 4$ distinct truth functions on single variable. These can be depicted in a truth table as follows:

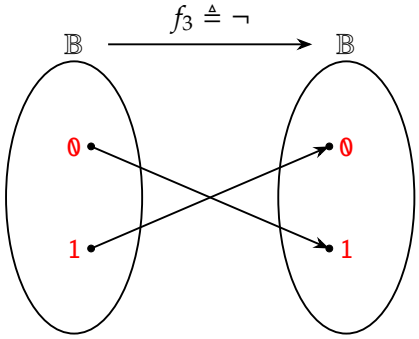
$P \in \mathbb{B}$	$\mathbb{0}$	1
f_1	$\mathbb{0}$	$\mathbb{0}$
f_2	$\mathbb{0}$	1
f_3	1	$\mathbb{0}$
f_4	1	1



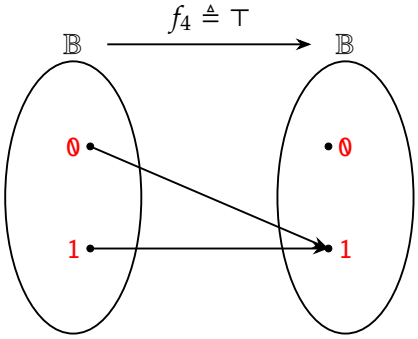
f_1 is the *contradiction function*.



f_2 is the *identity function*



f_3 is the *logical not function*



f_4 is the *tautology function*

□

Remark A.1.1 (Group Structure)**.**

- $(\{\mathrm{id}_P, \neg\}, \circ)$ is a group, but $(\{\mathrm{id}_P, \bot\}, \circ)$ and $(\{\mathrm{id}_P, \top\}, \circ)$ are not:

$\circ$	id_P	$\neg$
id_P	id_P	$\neg$
$\neg$	$\neg$	id_P

$\circ$	id_P	$\bot$
id_P	id_P	$\bot$
$\bot$	$\bot$	$\bot$

$\circ$	id_P	$\top$
id_P	id_P	$\top$
$\top$	$\top$	$\top$

Remark A.1.2. The structure is a non-associative magma, also known as a groupoid.

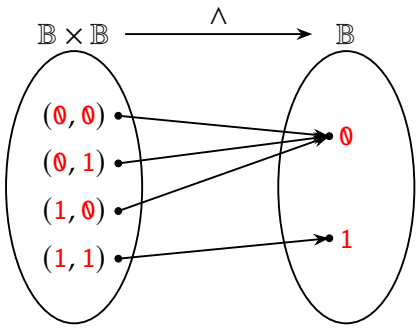
$\circ$	$\bot$	P	$\neg$	$\top$
$\bot$	$\bot$	$\bot$	$\top$	$\top$
P	$\bot$	P	$\neg$	$\top$
$\neg$	$\bot$	$\neg$	P	$\top$
$\top$	$\bot$	$\top$	$\bot$	$\top$

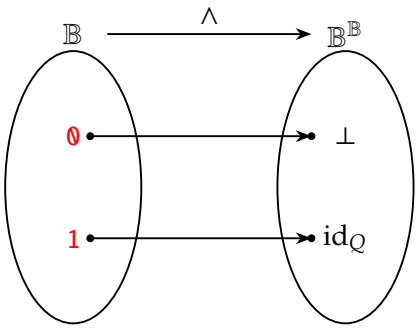
Binary Truth Functions

Corollary A.1.2 *There are 16 distinct binary truth functions:*

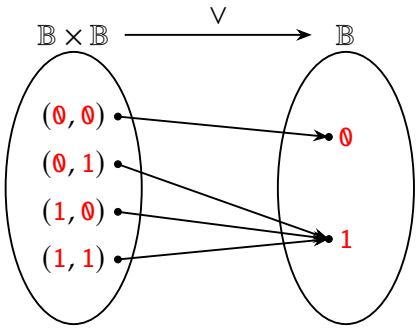
- *Two constant operations:*
  - $f_{\mathbb{1}}(p,q) = \mathbb{1}$
  - $f_{\mathbb{0}}(p,q) = \mathbb{0}$
- *Two projections:*
  - $\mathsf{Proj}_1(p,q) = p$
  - $\mathsf{Proj}_2(p,q) = q$
- *Two negated projections:*
  - $\overline{\mathsf{Proj}_1}(p,q) = \neg p$
  - $\overline{\mathsf{Proj}_2}(p,q) = \neg q$
- *The conjunction:* $p \wedge q$
- *The disjunction:* $p \vee q$
- *Two conditionals:*
  - $p \implies q$
  - $q \implies p$
- *The biconditional (iff):* $p \iff q$
- *The exclusive or (xor):* $\neg(p \iff q)$
- *Two negated conditionals:*
  - $\neg(p \implies q)$
  - $\neg(q \implies p)$
- *The NAND* $p \uparrow q$
- *The NOR* $p \downarrow q$

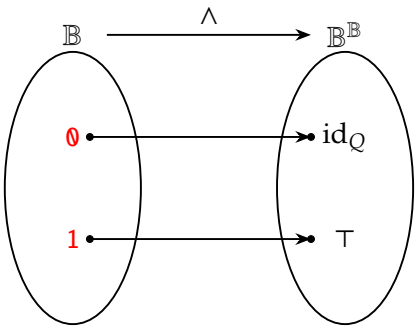
Proof. From Count of Truth Functions there are $2^{(2^2)} = 16$ distinct truth functions on 2 variable. These can be depicted in a truth table as follows:

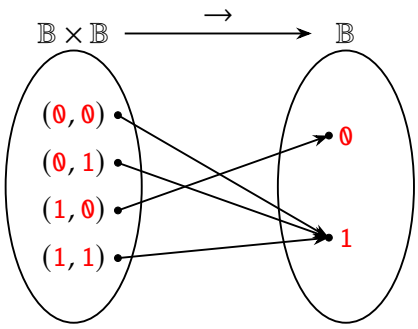


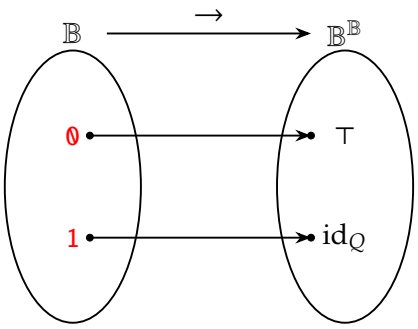


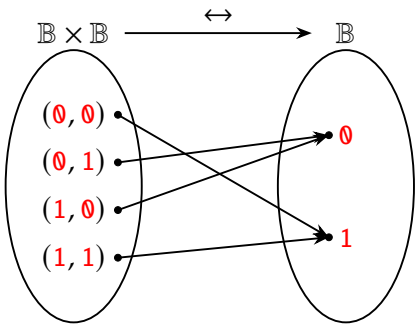
p	1	1	0	0
q	1	0	1	0
$f_0(p,q)$	0	0	0	0
$p \downarrow q$	0	0	0	1
$\neg(p \impliedby q)$	0	0	1	0
$\overline{\mathsf{Proj}_1}(p,q)$	0	0	1	1
$\neg(p \implies q)$	0	1	0	0
$\overline{\mathsf{Proj}_2}(p,q)$	0	1	0	1
$\neg(p \iff q)$	0	1	1	0
$p \uparrow q$	0	1	1	1
$p \wedge q$	1	0	0	0
$p \iff q$	1	0	0	1
$\mathsf{Proj}_2(p,q)$	1	0	1	0
$p \implies q$	1	0	1	1
$\mathsf{Proj}_1(p,q)$	1	1	0	0
$p \impliedby q$	1	1	0	1
$p \vee q$	1	1	1	0
$f_1(p,q)$	1	1	1	1

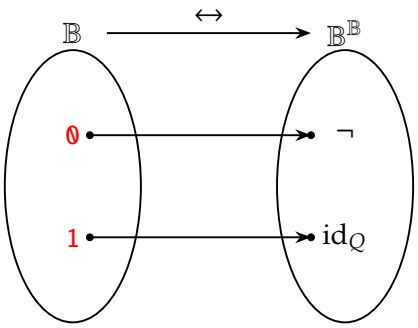












□

A.2 Backus-Naur Form (BNF)

Backus-Naur Form (BNF) is a notation technique for context-free grammars, often used to describe the syntax of languages used in computing. Introduced by John Backus and Peter Naur in the 1960s, BNF was initially developed to describe the syntax of the ALGOL 60 programming language.

A grammar defines a language by providing a set of production rules that describe how sentences in the language can be formed. BNF uses non-terminal symbols, terminal symbols, and production rules to define these grammars.

Non-terminal symbols are placeholders for patterns of terminal symbols that can be generated by applying production rules.

Terminal symbols are the actual symbols of the language's alphabet, and they appear in the strings generated by the grammar.

Production rules define how non-terminal symbols can be replaced with combinations of non-terminal and terminal symbols. BNF uses a specific syntax to describe these rules:

```
<rule-name> ::= <expression>
```

Extended BNF (EBNF) provides additional notation to simplify grammar definitions, such as optional elements and repetitions.

BNF is a powerful tool for defining the syntax of programming languages and has been fundamental in the development of many language specifications. For more information, refer to resources such as the *ALGOL 60 Report*, or textbooks on formal language theory and compiler design.

Example A.2.1. Here is an example of BNF describing simple arithmetic expressions:

```
<expression> ::= <term> | <term> "+" <expression>
<term> ::= <factor> | <factor> "*" <term>
<factor> ::= <number> | "(" <expression> ")"
<number> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Bibliography

[1] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 1: Introduction to software analysis (1). YouTube, 2024. Accessed: 2024-06-28.

[2] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 1: Introduction to software analysis (2). YouTube, 2024. Accessed: 2024-06-28.

[3] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 4: Propositional logic (1). YouTube, 2024. Accessed: 2024-06-28.

[4] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 4: Propositional logic (2). YouTube, 2024. Accessed: 2024-06-28.

[5] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 4: Propositional logic (3). YouTube, 2024. Accessed: 2024-06-28.

[6] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 5: Problem solving using smt solver (1). YouTube, 2024. Accessed: 2024-06-28.

[7] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 5: Problem solving using smt solver (2). YouTube, 2024. Accessed: 2024-06-28.

[8] 오학주[교수 / 컴퓨터학과]. Cose419 lecture 5: Problem solving using smt solver (3). YouTube, 2024. Accessed: 2024-06-28.

[9] bycho211. Course overview. Naver Blog, 2017. Accessed: 2024-06-28.