Discrete Maths

Sets

Power set: $Pow(X) = \{A : A \subseteq X\}$

Empty set: $\emptyset$

Set difference: $A \setminus B$

Set symmetric difference: $A \oplus B$

Set complement: A^c

Laws:

$A \cup A = A$	
$A \cap A = A$	Idempotence
$(A^c)^c = A$	Double complementation
$A \cap \emptyset = \emptyset$	Annihilation
$(A \cap B)^c = A^c \cup B^c$	
$(A \cup B)^c = A^c \cap B^c$	DeMorgan's Laws

Formal Languages

Alphabet: A finite, non-empty set Σ

Word: A finite string of symbols from Σ

Empty word: λ (sometimes ϵ)

Σ^*: Set of all words

Σ^+: Set of all nonempty words

Concatenation: $XY = \{xy : x \in X \wedge y \in Y\}$

Kleene star: X^*, the set of words that are made up by concatenating 0 or more words in X

Relations

Properties include:

$(x, x) \in R$	Reflexive
$(x, x) \notin R)$	Anti-reflexive
$(x, y) \in R \rightarrow (y, x) \in R$	Symmetric
$(x, y), (y, x) \in R \rightarrow x = y$	Anti-symmetric
$(x, y), (y, z) \in R \rightarrow (x, z) \in R$	Transitive

Functions

Composition: $g \circ f \equiv g(f(x))$

Recursion & Induction

Recursion

Consists of a basis (B) and recursive process (R).

A sequence/object/algorithm is recursively defined when (typically)

(B) some initial terms are specified, perhaps only the first one;

(R) later terms stated as functional expressions of the earlier terms.

Induction

Mathematical Induction

Base Case [B]: $P(a_1), P(a_2), \ldots, P(a_n)$ for some small set of examples $a_1 \ldots a_n$ (often $n = 1$)

Inductive Step [I]: A general rule showing that if $P(x)$ holds for some cases $x = x_1, \ldots, x_k$ then $P(y)$ holds for some new case y, constructed in some way from $x_1, \ldots, x_k$

Conclusion: Starting with $a_1 \ldots a_n$ and repeatedly applying the construction of y from existing values, we can eventually construct all values in the domain of interest

Structural Induction

Base Case [B]: The property holds for all minimal objects – objects that have no predecessors; they are usually very simple objects allowing immediate verification

Inductive Step [I]: for any given object, if the property in question holds for all its predecessors (‘smaller’ objects) then it holds for the object itself

Propositional Logic

Well Formed Formulas

Let $\textsc{Prop} = \{p, q, r \dots\}$

Let $\Sigma = \textsc{Prop} \cup \{\top, \bot, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)\}$

The **well formed formulas** (wffs) over $\textsc{Prop}$ is the smallest set of words over Σ such that:

- $\top, \bot$ and all elements of $\textsc{Prop}$ are wffs
- If φ is a wff, then $\neg\varphi$ is a wff
- If φ and ψ are wffs, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$ and $(\varphi \leftrightarrow \psi)$ are wffs

Valuations

A truth assignment (or model) is a function $v : \textsc{Prop} \rightarrow \mathbb{B}$

We can extend v to a function $[\![\cdot]\!]_v : \text{WFFs} \rightarrow \mathbb{B}$ recursively:

- $[\![\top]\!]_v = \text{true}$
- $[\![\bot]\!]_v = \text{false}$
- $[\![p]\!]_v = v(p)$
- $[\![\neg\varphi]\!]_v = ![\![\varphi]\!]_v$
- $[\![(\varphi \wedge \psi)]\!]_v = [\![\varphi]\!]_v \ \&\& \ [\![\psi]\!]_v$
- $[\![(\varphi \vee \psi)]\!]_v = [\![\varphi]\!]_v \ || \ [\![\psi]\!]_v$
- $[\![(\varphi \rightarrow \psi)]\!]_v = ![\![\varphi]\!]_v \ || \ [\![\psi]\!]_v$
- $[\![(\varphi \leftrightarrow \psi)]\!]_v = (![\![\varphi]\!]_v \ || \ [\![\psi]\!]_v) \ \&\& \ (![\![\psi]\!]_v \ || \ [\![\varphi]\!]_v)$

CNF & DNF

A propositional formula is in CNF (conjunctive normal form) if it has the form $\bigwedge_i c_i$, where each clause c_i is a disjunction of literals.

A propositional formula is in DNF (disjunctive normal form) if it has the form $\bigvee_i c_i$, where each clause c_i is a conjunction of literals.

For every boolean expression ϕ there exists an equivalent expression in conjunctive normal form and an equivalent expression in disjunctive normal form.

Converting

Push Negations Down using De Morgan's laws and the double negation rule

$$\begin{aligned}\neg(x \vee y) &\equiv \neg x \wedge \neg y\\ \neg(x \wedge y) &\equiv \neg x \vee \neg y\\ \neg\neg x &\equiv x\end{aligned}$$

Using the **distribution rules**

$$\begin{aligned}x \vee (y_1 \wedge \cdots \wedge y_n) &= (x \vee y_1) \wedge \cdots \wedge (x \vee y_n)\\ (y_1 \wedge \cdots \wedge y_n) \vee x &= (y_1 \vee x) \wedge \cdots \wedge (y_n \vee x)\end{aligned}$$

Using the equivalence $A \rightarrow B \equiv \neg A \vee B$

Examples

CNF: $(p \vee \neg q) \wedge (u \vee v)$

DNF: $(p \wedge \neg q) \vee (u \wedge v)$

Predicate Logic

Domain Of Discourse

Predicates	Relations on the domain
Functions	Operators on the domain
Constants	"Name" elements of the domain
Variables	"Unnamed" elements of the domain (placeholders for elements)
Quantifiers	Range over domain elements

Vocabulary

A vocabulary indicates what predicates, functions and constants we can use to build up our formulas.

A vocabulary is a set of:

- Predicate "symbols" $P, Q, \ldots,$ each with an assoicated arity (number of arguments)
- Function "symbols" $f, g, \ldots,$ each with an assoicated arity (number of arguments)
- Constant "symbols" $c, d, \ldots$ (also known as 0-arity functions)

Natural Deduction

Below some of the natural deduction inference rules for propositional and predicate logic.

Note: the notation $[A] \ldots B$, means "assuming A, we can deduce B".

$\dfrac{A \qquad B}{A \wedge B}$ ($\wedge$-I)	$\dfrac{A}{A \vee B}$ ($\vee$-I1)	$\dfrac{B}{A \vee B}$ ($\vee$-I2)
$\dfrac{A \wedge B}{A}$ ($\wedge$-E1)	$\dfrac{A \wedge B}{B}$ ($\wedge$-E2)	$\dfrac{A \qquad \neg A}{\bot}$ ($\neg$-E)
$\dfrac{A \to B \qquad A}{B}$ ($\to$-E)	$\dfrac{A \leftrightarrow B \qquad A}{B}$ ($\leftrightarrow$-E1)	$\dfrac{A \leftrightarrow B \qquad B}{A}$ ($\leftrightarrow$-E2)
$\dfrac{A \vee B \qquad [A] \ldots C \qquad [B] \ldots C}{C}$ ($\vee$-E)	$\dfrac{[A] \ldots B}{A \to B}$ ($\to$-I)	$\dfrac{[A] \ldots B \qquad [B] \ldots A}{A \leftrightarrow B}$ ($\leftrightarrow$-I)
$\dfrac{[A] \ldots \bot}{\neg A}$ ($\neg$-I)	$\dfrac{[\neg A] \ldots \bot}{A}$ (IP)	$\dfrac{\bot}{A}$ (X)
$\dfrac{\neg\neg A}{A}$ (DNE)	$\dfrac{A}{A}$ (R)	$\dfrac{[A] \ldots B \qquad [\neg A] \ldots B}{B}$ (LEM)
$\dfrac{A \vee B \qquad \neg A}{B}$ (DS)	$\dfrac{A \to B \qquad \neg B}{\neg A}$ (MT)	$\dfrac{\neg(A \vee B)}{\neg A \wedge \neg B}$ (DM)
$\dfrac{}{a = a}$ (=-I)	$\dfrac{a = b \qquad A(a)}{A(b)}$ (=-E1)	$\dfrac{a = b \qquad A(b)}{A(a)}$ (=-E2)
$\dfrac{A(c) \qquad {\scriptstyle (1,\ 2,\ 3)}}{\forall x A(x)}$ ($\forall$-I)	$\dfrac{A(c) \qquad {\scriptstyle (2)}}{\exists x A(x)}$ ($\exists$-I)	$\dfrac{\forall x A(x)}{A(c)}$ ($\forall$-E)
$\dfrac{\exists x A(x) \qquad [A(c)] \ldots B \qquad {\scriptstyle (1,\ 2,\ 4)}}{B}$ ($\exists$-E)		(1): c is arbitrary (2): x is not free in $A(c)$ (3): c is not free in $A(x)$ (4): c is not free in B

Hoare Logic

$\mathcal{L}$

Assignment	$x := e$
Sequencing	$P; Q$
Conditional	**if** b **then** P **else** Q **fi**
While	**while** b **do** P **od**

Hoare Triples

$\{\varphi\}\ P\ \{\psi\}$ where

φ: The **precondition** – an assertion about the state prior to the execution of the code fragment

P: The **code fragment**

ψ: The **postcondition** – an assertion about the state after to the execution of the code fragment if it terminates

$\mathcal{L}$ Rules

Assignment
$$\frac{}{\{\varphi[e/x]\}x := e\{\varphi\}} \quad \text{(ass)}$$

Sequence
$$\frac{\{\varphi\}\ P\ \{\psi\} \qquad \{\psi\}\ Q\ \{\rho\}}{\{\varphi\}\ P; Q\ \{\rho\}} \quad \text{(seq)}$$

Conditional
$$\frac{\{\varphi \land g\}\ P\ \{\psi\} \qquad \{\varphi \land \neg g\}\ Q\ \{\psi\}}{\{\varphi\}\ \textbf{if}\ g\ \textbf{then}\ P\ \textbf{else}\ Q\ \textbf{fi}\ \{\psi\}} \quad \text{(if)}$$

While
$$\frac{\{\varphi \land g\}\ P\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ g\ \textbf{do}\ P\ \textbf{od}\ \{\varphi \land \neg g\}} \quad \text{(loop)}$$

Precondition Strengthening & Postcondition Weakening
$$\frac{\varphi' \to \varphi \qquad \{\varphi\}\ P\ \{\psi\} \qquad \psi \to \psi'}{\{\varphi'\}\ P\ \{\psi'\}} \quad \text{(cons)}$$

$\mathcal{L}$ Semantics

An **environment** or **state** is a function from variables to numeric values. We denote by $\textsc{Env}$ the set of all environments

Assignment: $(\eta, \eta') \in [\![x := e]\!]$ iff $\eta' = \eta[x \mapsto [\![e]\!]^\eta]$

Sequencing: $[\![P; Q]\!] = [\![P]\!]; [\![Q]\!]$

Conditional: $[\![\textbf{if } b \textbf{ then } P \textbf{ else } Q \textbf{ fi}]\!] = [\![b; P]\!] \cup [\![\neg b; Q]\!]$

While: $[\![\textbf{while } b \textbf{ do } P \textbf{ od}]\!] = [\![b; P]\!]^*; [\![\neg b]\!]$

Weakest Precondition

Given a program P and a postcondition ψ, the weakest precondition of P with respect to ψ, $wp(P, \psi)$, is a predicate φ such that

$$\{\varphi\}\ P\ \{\psi\} \quad \text{and} \quad \text{if } \{\varphi'\}\ P\ \{\psi\} \text{ then } \varphi' \to \varphi$$

Assignment: $wp(x := e, \psi) = \psi[e/x]$

Sequence: $wp(P; S, \psi) = wp(P, wp(S, \psi))$

Conditional: $wp(\textbf{if } \mathrm{b} \textbf{ then } \mathrm{P} \textbf{ else } \mathrm{Q} \textbf{ fi}, \psi) = (b \wedge wp(P, \psi)) \vee (\neg b \wedge wp(Q, \psi))$

While:

Find a loop invariant I such that

- $\varphi \to I$
- $\{I \wedge b\}\ P\ \{I\}$
- $I \wedge \neg b \to \psi$

Termination

$[\varphi]P[\psi]$ asserts that if φ hold at a starting state and P is executed, then P will terminate and ψ will hold in the resulting state.

Rules For Total Correctness

Assignment	$$\frac{}{[\varphi[e/x]]x := e[\varphi]}$$	(ass)
Sequence	$$\frac{[\varphi]\ P\ [\psi] \qquad [\psi]\ Q\ [\rho]}{[\varphi]\ P;Q\ [\rho]}$$	(seq)
Conditional	$$\frac{[\varphi \wedge g]\ P\ [\psi] \qquad [\varphi \wedge \neg g]\ Q\ [\psi]}{[\varphi]\ \textbf{if}\ g\ \textbf{then}\ P\ \textbf{else}\ Q\ \textbf{fi}\ [\psi]}$$	(if)
While	$$\frac{[\varphi \wedge g \wedge (v = N)]\ P\ [\varphi \wedge (v < N)] \qquad (\varphi \wedge g) \to (v > 0)}{[\varphi]\ \textbf{while}\ g\ \textbf{do}\ P\ \textbf{od}\ [\varphi \wedge \neg g]}$$	(loop)
Precondition Strengthening & Postcondition Weakening	$$\frac{\varphi' \to \varphi \qquad [\varphi]\ P\ [\psi] \qquad \psi \to \psi'}{[\varphi']\ P\ [\psi']}$$	(cons)

Terminating While Loops

Partial correctness: Find an invariant I such that:

- $\varphi \to I$
- $[I \wedge b]\ P\ [I]$
- $(I \wedge \neg b) \to \psi$

Show termination: Find a variant v such that:

- $(I \wedge b) \to v > 0$
- $[I \wedge b \wedge v = N]\ P\ [v < N]$

$\mathcal{L}^+$

Assignment	$x := e$	
Predicate	φ	
Sequencing	$P;Q$	
Choice	$P + Q$	Make a non-deterministic choice between P and Q
Loop	P^*	Loop for a non-deterministic number of iterations

State Machines

A **transition system** is a pair $(S, \rightarrow)$ where:

- S is a set of states
- $\rightarrow \subseteq S \times S$ is a transition relation

If $(s, s') \in \rightarrow$, we write $s \rightarrow s'$.

- S may have a start state s_0
- S may have final states $F \subseteq S$
- The transitions may be labelled by elements of a set Λ:
  - $\rightarrow \subseteq S \times \Lambda \times S$
  - $(s, a, s') \in \rightarrow$ is written as $s \xrightarrow{a} s'$
- If $\rightarrow$ is a function, we say the system is deterministic, otherwise it is non- deterministic

Runs & Reachability

Given a transition system $(S, \rightarrow)$ and states $s, s' \in S$

- A **run** from s is a (possibly infinite) sequence $s_1, s_2 \ldots$ such that $s = s_1$ and $s_i \rightarrow s_{i+1}$ for all $i \geq 1$
- We say s' is **reachable** from s, written $s \xrightarrow{*} s'$ if (s, s') is in the transitive closure of $\rightarrow$
- A state s' is reachable from s if there is a run from s which contains s'

Safety & Liveness

Safety: Something bad will never happen. In the context of transition systems, "will a transition system always avoid a particular state or states"

Liveness: Something good will happen. In the context of transition systems, "will a transition system always reach a particular state or states"

The Invariant Principle

A **preserved invariant** of a transition system is a unary predicate φ on states such that if $\varphi(s)$ holds, and $s \to s'$, then $\varphi(s')$ holds.

Invariant Principle: If a preserved invariant holds at a state s, then it holds for all states reachable from s

Partial & Total Correctness, & Termination

Partial Correctness: A transition system is partially correct for φ if $\varphi(s')$ holds for all states $s' \in F$ that are reachable from s_0

Termination: A transition system terminates from a state $s \in S$ if there is an $N \in \mathbb{N}$ such that all runs from s have lenght at most N

Total Correctness: A transition system is totally correct for φ if it terminates from s_0 and φ holds in the last state of every run

Finite State Automata

Deterministic Finite Automata

A DFA is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is the input alphabet
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states

For a DFA $A = (Q, \Sigma, \delta, q_0, F)$, the **language** of A, $L(A)$ is the set of words from Σ^* which are accepted by A.

Non-Deterministic Finite Automata

A NFA is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is the input alphabet
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states

For any DFA, there exists an NFA, and vice versa.

Regular Languages

A language $L \subseteq \Sigma^*$ is **regular** is there is some DFA A, such that $L = L(A)$.

Equivalently, there is some NFA B such that $L = L(B)$.

Complementation: If L is a regular language then $L^c = \Sigma^* \setminus L$ is a regular language

Union: If L_1 and L_2 are regular languages, then $L_1 \cup L_2$ is regular

Intersection: If L_1 and L_2 are regular languages, then $L_1 \cap L_2$ is regular

Concatenation: If L_1 and L_2 are regular languages, then $L_1 \cdot L_2$ is regular

Kleene star: If L is a regular language, then L^* is regular

Regular Expressions

- $\emptyset$ is a regular expression
- ϵ is a regular expression
- a is a regular expression for all $a \in \Sigma$
- If E_1 and E_2 are regular expressions, then $E_1 E_2$ is a regular expression
- If E_1 and E_2 are regular expressions, then $E_1 + E_2$ is a regular expression
- If E is a regular expression, then E^* is a regular expression

Kleene's theorem

- For any regular expression E, $L(E)$ is a regular language
- For any regular language L, there is a regular expression E such that $L = L(E)$

Myhill-Nerode Theorem

Let $x, y \in \Sigma^*$ and let $L \subseteq \Sigma^*$. We say that x and y are $L-$indistinguishable, written $x \equiv_L y$ if for every $z \in \Sigma^*$:

$$xz \in L \text{ iff } yz \in L$$

$\equiv_L$ is an equivalence relation.

The index of L is the number of equivalence classes of $\equiv_L$.

The index of L may be finite or infinite.

Myhill-Nerode Theorem: L is regular if and only if L has finite index. Moreover, the index is the size (= number of states) of the smallest DFA accepting L.

Context Free Grammars

A context free grammar is a 4-tuple $G = (V, \Sigma, R, S)$ where

- V is a finite set of variables (or non-terminals)
- Σ (the alphabet) is a finite set of terminals
- R is a finite set of productions. A production (or rule) is an element of $V \times (V \cup \Sigma)^*$, written $A \to w$
- $S \in V$ is the start symbol