Final Year Project– Equivalence Preserving Program Transformations for Program Verification

SEMESTER I-II, 2023-2024

Lecture Notes

Written by: Lua Yi Da

- This is a set of notes written to demonstrate understanding of the prerequisites of the upcoming FYP in program verification.
- It encompasses the following topics/papers: **Automata theory, Decidable Verification of Uninterpreted Programs, Calculus of Computation, Automated Hypersafety Verification, Tree Automata**.
- Each section contains the most important definitions and theorems for the relevant topic/paper, as well as the corresponding proofs and accompanying details.

Regular Languages

Definition 1 (Finite automaton)**.** A finite automaton is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \longrightarrow Q$ is the transition function, $q_o \in Q$ is the start state and $F \subseteq Q$ is the set of accept states.

We say that the finite automaton M accepts the string w if there exists a sequence of states $r_o, \ldots, r_n \in Q$ s.t.:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}, 0 \leq i \leq n-1$
3. $r_n \in F$

Definition 2 (Regular language)**.** A language L is regular if some DFA recognises it i.e. L = $\{w \in \Sigma^* \mid \text{M accepts w}\}$.

We now define some basic set operations for languages. Given languages A and B:

1. $A \cup B = \{x \in \Sigma^* \mid x \in A \text{ or } x \in B\}$ (Union)
2. $A \circ B = \{xy \in \Sigma^* \times \Sigma^* \mid x \in A,\ y \in B\}$ (Concatenation)
3. $A^* = \{x_1 x_2 \ldots x_k \in \Sigma^* \times \cdots \times \Sigma^* \mid k \geq 0,\ x_i \in A,\ 1 \leq i \leq k\}$ (Star)

Definition 3 (Nondeterministic finite automaton)**.** A nondeterministic finite automaton is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma_\epsilon \longrightarrow \mathscr{P}(Q)$ is the transition function, $q_o \in Q$ is the start state and $F \subseteq Q$ is the set of accept states.

We say that the NFA N accepts the string w if $w = y_1 \ldots y_m, y_i \in \Sigma_\epsilon$, $1 \leq i \leq m$ and there exists a sequence of states $r_o, \ldots, r_m \in Q$ s.t.:

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1}), 0 \leq i \leq m-1$
3. $r_m \in F$

2 machines are said to be equivalent if they recognise the same language.

Nondeterminism can be viewed as a kind of parallel computation, where computation accepts if any branch accepts.

Theorem 1 (Equivalence of DFA and NFA)**.** Every NFA has an equivalent DFA.

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be NFA recognising the language A.

We construct the DFA $D = (Q', \Sigma, \delta', q_0', F')$.

For any $R \subseteq Q$, define $E(R) = \{q \mid$ q can be reached from R by travelling along ≥ 0 ϵ arrows$\}$.

Set $Q' = \mathscr{P}(Q), \delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a)), q_0' = E(\{q_0\}), F' = \{R \in Q' \mid$ R contains accept state of N$\}$. Clearly, every computation step of D on an input enters a state corresponding to the subset of states that N could be in at that point. Trivially, D recognises A. □

Theorem 2. A language is regular $\iff$ There exists an NFA recognising the language.

Proof. ($\implies$) A language is regular if some DFA D recognises it. By construction, any DFA is also an NFA.

($\impliedby$) Suppose an NFA recognises the language A. By Theorem 1, it follows that there exists an equivalent DFA recognising the language A, in turn implying regularity of A. □

Theorem 3. The class of regular languages is closed under union.

Proof. Suppose $D_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognises A_1 and $D_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ recognises A_2.

We construct a DFA $D = (Q, \Sigma, \delta, q_0, F)$ to recognise $A \cup B$.

Set $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$, $q_0 = (q_1, q_2)$, $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. □

Theorem 4. The class of regular languages is closed under concatenation.

Proof. Suppose $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognises A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognises A_2.

We construct a NFA $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognise $A \circ B$.

Set $Q = Q_1 \cup Q_2$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a), q \in Q_1 \land q \notin F_1 \\ \delta_1(q, a), q \in F_1 \land a \neq \epsilon \\ \delta_1(q, a) \cup \{q_2\}, q \in F_1 \land a = \epsilon \\ \delta_2(q, a), q \in Q_2 \end{cases}$$

□

Theorem 5. The class of regular languages is closed under the star operation.

Proof. Suppose $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognises A.

We construct a NFA $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognise A^*.

Set $Q = Q_1 \cup \{q_0\}$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a), q \in Q_1 \wedge q \notin F_1 \\ \delta_1(q,a), q \in F_1 \wedge a \neq \epsilon \\ \delta_1(q,a) \cup \{q_1\}, q \in F_1 \wedge a = \epsilon \\ \{q_1\}, q = q_0 \wedge a = \epsilon \\ \varnothing, q = q_0 \wedge a \neq \epsilon \end{cases}$$

□

Definition 4 (Regular Expression)**.** R is a regular expression if R is:

1. a, for some $a \in \Sigma$
2. ϵ
3. $\varnothing$
4. $(R_1 \cup R_2)$, R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$, R_1 and R_2 are regular expressions
6. (R_1^*), R_1 is a regular expression

Definition 5 (Generalised NFA)**.** A generalised NFA is a tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where Q is a finite set of states, Σ is a finite input alphabet, $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_start\}) \longrightarrow R$ (where R is the set of regular expressions) is the transition function, q_{start} is the start state, q_accept is the accept state.

We say that GNFA G accepts the string w if $w = w_1 \dots w_k, w_i \in \Sigma^*, 1 \leq i \leq k$, and there exists a sequence of states $q_0, \dots, q_k \in Q$ st.:

1. $q_0 = q_{start}$
2. $q_k = q_{accept}$
3. $\forall i$, $w_i \in L(R_i)$, $R_i = \delta(q_{i-1}, q_i)$ i.e. R_i is the expression on the arrow from q_{i-1} to q_i.

Theorem 6. A language is regular $\iff$ There exists some regular expression describing it.

Theorem 7. If A is a regular language, then there exists a number p (called the pumping length) where if s is a string in A of length at least p, then $s = xyz$ satisfying:

1. $\forall i \geq 0$, $xy^iz \in A$
2. $|y| > 0$

3. $|xy| \leq p$

Proof. Let $M = (Q, \Sigma, \delta, q_1, F)$ be DFA recognising A, and let p be the number of states of M.

Let $s = s_1 \ldots s_n, n \geq p$ be a string in A. Let $r_1, \ldots, r_{n+1}$ be the sequence of states that M enters while processing s i.e. $r_{i+1} = \delta(r_i, s_i), 1 \leq i \leq n$. Note that this sequence has length $n + 1 \geq p + 1$. For the first $(p+1)$ elements in the sequence, at least 2 of them must be the same state by the pigeonhole principle. Call the first r_j and the second $r_l, l \leq p + 1$.

Let $x = s_1 \ldots s_{j-1}, y = s_j \ldots s_{l-1}, z = s_l \ldots s_n$. By construction, M trivially accepts $xy^i z, i \geq 0$.

Further, notice that $j \neq l \implies |y| > 0$ and $l \leq p + 1 \implies |xy| \leq p$. □

Context-Free Languages

Definition 6 (Context-Free Grammar)**.** A context-free grammar is a tuple (V, Σ, R, S) where V is a finite set of variables, Σ is a finite set (disjoint from V) of terminals, R is a finite set of rules (where a rule is a variable + a string of variables and terminals), $S \in V$ is the start variable.

We say that $u \overset{*}{\Longrightarrow} v$ if $u = v$ or there exists $u_1, \ldots, u_k, k \geq 0$ and $u \Longrightarrow u_1 \Longrightarrow \cdots \Longrightarrow u_k \Longrightarrow v$. The language of a context free grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Longrightarrow} w\}$.

Definition 7 (Ambiguity)**.** A string w is derived ambiguously in a context-free grammar G if it has 2 or more different leftmost derivations. A grammar G is ambiguous if it derives some string ambiguously.

Definition 8 (Chomsky Normal Form)**.** A context-free grammar is in Chomsky Normal Form if every rule is of the following forms:

1. $A \longrightarrow BC$
2. $A \longrightarrow a$

Here, $a \in \Sigma, A \in V, B, C \in V - S$. Additionally, we permit the rule $S \longrightarrow \epsilon$.

Theorem 8. Any context-free language is generated by a context-free grammar in Chomsky Normal Form.

Proof. We want to show that any context-free grammar can be converted into an equivalent grammar in Chomsky Normal Form.

First, we add a new start variable S_0 and the rule $S_0 \longrightarrow S$ to guarantee that the start variable S does not appear in RHS of a rule.

To deal with ϵ-rules, we consider the following procedure. Remove an ϵ-rule $A \longrightarrow \epsilon, A \in V - S$. Then for each occurrence of an A in RHS of rule, we add a new rule that deletes the occurrence of A i.e. Replace $R \longrightarrow uAv$ with $R \longrightarrow uv$. If we have the rule $R \longrightarrow A$, we replace it with $R \longrightarrow \epsilon$. Repeat this procedure till all ϵ-rules not involving S are removed.

To deal with unit rules, consider the following. Remove a unit rule $A \longrightarrow B$. Then, whenever a rule $B \longrightarrow u$ appears, we add the rule $A \longrightarrow u$ unless such a rule was previously removed. Repeat this procedure till all unit rules are removed.

Finally, we convert this new set of rules into Chomsky Normal Form. For each rule $A \longrightarrow u_1 \ldots u_k, k \geq 3$ and each u_i is a variable/terminal symbol, with the rules $A \longrightarrow u_1 A_1$, $A_1 \longrightarrow u_2 A_2$, $\ldots$, $A_{k-2} \longrightarrow u_{k-1} u_k$, where each A_i is a new variable. Finally, we replace any terminal u_i with the rule $U_i \longrightarrow u_i$. $\square$

Definition 9 (Pushdown Automaton)**.** A pushdown automaton is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite stack alphabet, $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow \mathscr{P}(Q \times \Gamma_\epsilon)$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accept states.

We say that the pushdown automaton M accepts the string w if $w = w_1 \dots w_m, w_i \in \Sigma_\epsilon, 1 \leq i \leq m$ and there exists a sequence of states $r_0, \dots, r_m \in Q$ and $s_0, \dots, s_m \in \Gamma^*$ st.:

1. $r_0 = q_0, s_0 = \epsilon$
2. For $0 \leq i \leq m-1$, $(r_i, b) \in \delta(r_i, w_{i+1}, a), s_{i+1} = at, s_{i+1} = bt$ for $a, b \in \Gamma_\epsilon, t \in \Gamma^*$
3. $r_m \in F$

Theorem 9. A language is context-free $\iff$ Some pushdown automaton recognises the language.

Proof. Check Page 118 of Sipser's Introduction to the Theory of Computation for full details on proof. □

Theorem 10. Every regular language is context-free.

Proof. Note that a finite automaton can be equivalently formulated as a pushdown automaton that ignores the stack. This trivially gives the required construction. □

Theorem 11. If A is a context-free language, then there exists a number p (called the pumping length) where, if s is a string in A of length at least p, then $s = uvxyz$ satisfying:

1. For each $i \geq 0$, $uv^ixy^iz \in A$
2. $|vy| > 0$
3. $|vxy| \leq p$

Proof. Check Page 126 of Sipser's Introduction to the Theory of Computation for full details on proof. □

Definition 10 (Deterministic PDA)**.** A deterministic pushdown automaton is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite stack alphabet, $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow (Q \times \Gamma_\epsilon) \cup \{\varnothing\}$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accept states.

Further, δ must satisfy the following: For every $q \in Q, a \in \Sigma, x \in \Gamma$, exactly one of $\delta(q, a, x), \delta(q, a, \epsilon), \delta(q, \epsilon, x), \delta(q, \epsilon, \epsilon)$ is not $\varnothing$. The language of a deterministic PDA is called a DCFL (deterministic context-free language).

Theorem 12. Every DPDA has an equivalent DPDA that always reads the entire input string.

Theorem 13. The class of DCFLs is closed under complementation.

Definition 11 (Endmarked language)**.** For any language A, the endmarked language adds a special endmarker symbol to mark where an input string ends. i.e. $A \dashv = \{w \dashv | w \in A\}$.

Theorem 14. A is a DCFL $\iff$ $A \dashv$ is a DCFL.

Proof. ($\implies$): Suppose a DPDA P recognises the language A. Then DPDA P' recognises $A \dashv$ by simulating P until P' reads $\dashv$. P' terminates and enters accept state if P had entered an accept state during the previous symbol.

($\impliedby$): Suppose a DPDA P recognises $A \dashv$, we want to construct a DPDA P' recognising A. As P' reads every input symbol,P' determines whether P would accept if that symbol were $\dashv$. If so, P' enters accept state. After reading $\dashv$, P' can still operate the stack, so we store additional information on the stack to allow P' to immediately determine if P accepts. This additional information indicates from which states P would eventually acceept while manipulating the stack, but without reading further inputs. For details, refer to Sipser's Introduction to the Theory of Computation Page 134. $\square$

There are still several missing details in this section, particularly in relation to DCFLs and DCFGs. Revisit!