Vorbereitung für die Theorieprüfung Effiziente Algorithmen, Kryptographie und Angewandte Automatentheorie

Dominique Ziegelmayer

dominique.ziegelmayer@rwth-aachen.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

Liebe Kommilitonen, in diesem Dokument habe ich die 120 häufigsten Prüfungsfragen für die Diplomprüfung der Theoretischen Informatik in den Gebieten Effiziente Algorithmen, Kryptographie und Angewandte Automatentheorie zusammengefasst und mich bemüht zu fast allen Fragen ein paar Worte zu schreiben. Ich möchte hier ausdrücklich darauf hinweisen, dass ich keinerlei Garantie auf Vollständigkeit und Richtigkeit gebe. Dieses Dokument soll eine Hilfe für all diejenigen sein, die kurz vor Ihrer Prüfung stehen und selbst ihr Wissen überprüfen möchten. Nicht empfehlenswert ist ausschließlich aus diesem Dokument zu lernen und keinerlei Hintergründe zu kennen. Verbesserungen und Anmerkungen könnt ihr mir gerne schicken, allerdings muss ich natürlich sehen, wieviel Zeit ich habe, diese zu berücksichtigen. Ansonsten allen viel Erfolg, Dominique Ziegelmayer.

Kapitel 2

Effiziente Algorithmen

2.1 Reduktionen

2.1.1 Was ist NP-Vollständigkeit?

Eine Sprache L_0 ist NP Vollständig, wenn

- $L_0 \in NP$
- $L \leq_p L_0$ für alle $L \in NP$ (L_O NPSchwer)

2.1.2 Welche Polynomzeit-Reduktionen kennen Sie?

SAT $\leq_p$ CLIQUE, CLIQUE $\leq_p$ VC, 3-SAT $\leq_p$ SOL-0/1-LP

2.1.3 Erklären Sie SAT $\leq_p$ CLIQUE

2.1.4 Erklären Sie CLIQUE $\leq_p$ VC

2.1.5 Erklären Sie 3-SAT $\leq_p$ SOL-0/1-LP

2.1.6 Erklären Sie HC $\leq_p$ RHC

2.2 Techniken

2.2.1 Was ist Divide-and-Conquer?

Divide-and-Conquer ist eine Design-Technik, bei der man das Problem in Subprobleme teilt und diese wieder teilt und so weiter, bis das Problem leicht zu lösen ist. Es handelt sich hierbei um einen Top-Down Ansatz.

2.2.2 Welche Algorithmen für Divide and Conquer kennen Sie?

Multiplikation großer Zahlen, Parametrisierter Algorithmus für VC und Algorithmus für 3-Sat in $O(1,84^n)$

2.2.3 Erklären Sie die Multiplikation großer Zahlen

Seien $a = a_n a_{n-1}...a_1$ und $a = a_n a_{n-1}...a_1$ zwei große Binärzahlen.
Bilde $A = Number(a_n a_{n-1}...a_1) = \underbrace{Number(a_n...a_{n/2+1})}_{A_1} \cdot 2^{n/2} + \underbrace{Number(a_{n/2}...a_1)}_{A_2}$.

Seien B, B_1, B_2 analog definiert. Dann Gilt:
$AB = A_1B_1 \cdot 2^n + (A_1B_2 + A_2B_1) \cdot 2^{n/2} + A_2B_2$
Analysiert man hier die Komplexität, erhält man die folgende Rekursionsgleichung: $Time(n) = 4 \cdot Time(n/2) + cn$, Da der konstante Aufwand der Additionen und Shifts cn ist und der Algorithmus insgesamt 4 mal mit Subproblemen (Multiplikationen) der größe max. $(n/2)$ wieder aufgerufen werden muss. Bei der Auflösung der Rekursionsgleichung erhält man hier $O(n^2)$. Da dies keine Verbesserung zum Schulbuchalgorithmus ist, minimiert man die Anzahl der Multiplikationen:
$(A_1B_2 + A_2B_1) = (A_1 - A_2)(B_2 - B1) \cdot A_1B_1 + A_2B_2$
Durch die eine gesparte Multiplikation erhält man:
$Time(n) = 3 \cdot Time(n/2) + cn \Rightarrow Time(n) \in O(n^{1,59})$

2.2.4 Was ist Dynamisches Programmieren?

Dynamisches Programmieren ist eine Design-Technik, bei der man mit der Lösung der kleinsten Teilprobleme beginnt und diese dann zur Gesamtlösung zusammensetzt. Es handelt sich hierbei um einen Bottom-Up Ansatz.

2.2.5 Welche Algorithmen für Dynamisches Programmieren kennen Sie?

Floyd-Algorithmus, DPKP (Pseudo-Polynomialzeit)

2.2.6 Erklären Sie den Floyd-Algorithmus

Der Floyd Algorithmus dient zur Berechnung der kürzesten Pfade zwischen je zwei Knoten v_i und v_j. Dazu berechnet er für $k = 0, ..., n$ jeweils $cost_k(i, j)$, was den Kosten des kürzesten Pfades zwischen v_i und v_j entspricht, wobei dieser über die internen Knoten $(v_1, ..., v_k)$ geht.

2.2.7 Was ist Lokale Suche?

Lokale Suche ist ein Design-Konzept, dass auf der Idee von Nachbarschaften beruht. Eine Nachbarschaft ist eine Abbildung $f_x : M(x) \rightarrow Pot(M(x))$ die Reflexiv und Symmetrisch ist und es von jeder Lösung in $M(x)$ einen Pfad zu jeder anderen Lösung über Nachbarschaften gibt. Eine Nachbarschft wird in der Regel durch eine lokale Transformation, z.B. das Tauschen einer Kante in einem Graphenproblem oder das flippen eines Bits in einem Erfüllbarkeitsproblem, definiert.

2.2.8 Was gibt es für gute, bzw. schlechte Beispiele für lokale Suche?

Beispiel für gute Lösung: MST, Beispiel für schlechte Lösung: TSP (Pathologischer Fall)

2.2.9 Erklären Sie die lokale Suche für MST

Beim Minimum-Spanning-Tree Problem wird die Nachbarschaft definiert, als die Lösungen, die aus $\alpha \in M(x)$ entstehen, indem man eine Kante hinzufügt (Kreis) und dann eine Kante des Kreises entfernt, so dass die Spannbaum-Eigenschaft wiederhergestellt ist. Bis keine Verbesserung mehr möglich ist, verfolgt der Algorithmus dabei folgendes Rezept: Füge jeweils die günstigste Kante hinzu und Streiche die teuerste Kante aus dem entstandenen Kreis heraus. Der Algorithmus liefert die optimale Lösung, da die Nachbarschaft exakt und polynomzeitdurchsuchbar ist.

2.2.10 Was ist das Pathologische TSP?

Das pathologische TSP ist eine Instanz des TSP, so dass die Lokale Suche beliebig schlechte Ergebnisse liefert.
Hierbei existiert:

- genau eine optimale Lösung mit Kosten $8k$
- $2^{k-1}(k-1)!$ zwitbeste Lösungen mit Kosten $2^{8k} + 5k$
- Jede zweitbeste Lösung ist ein lokales Optimum bezüglich einer $(3k-1)$-Exchange-Neighbourhood.

Das Pathologische TSP ist ein vollständiger Graph, der aus k Diamonds besteht. (Kantenkosten und Beweis bitte dem Buch entnehmen).

2.2.11 Erklären Sie Kerningham-Lin Variable Deep-Search

Dieser Algorithmus ist eine Kombination aus lokaler Suche und Greedy Strategie. Der Algorithmus läuft wie folgt ab:

1. Bilde eine mögliche Lösung $\alpha \in M(x)$ mit $\alpha = (p_1 p_2 ... p_n)$ und eine Menge $EXCHANGE = \{1, ..., n\}$
2. Suche nach der besten Lösung $\beta \in Neigh(\alpha)$, die sich nur in den Indizes aus $EXCHANGE$ von α unterscheidet. Streiche dann die Indizes aus $EXCHANGE$ in denen sich α und β unterscheiden.
3. Suche nach der besten Lösung $\gamma \in Neigh(\beta)$, die sich nur in den Indizes aus $EXCHANGE$ von α unterscheidet. Streiche dann die Indizes aus $EXCHANGE$ in denen sich α und β unterscheiden. Gehe so weiter vor, bis $EXCHANGE$ leer ist.
4. Bestimme die beste Lösung $\tau \in \{\alpha, \beta, \gamma, ...\}$
5. Prüfe, ob τ besser ist als α. Wenn ja setze $\alpha = \tau$ und starte den Algorithmus erneut. Wenn nein, gebe α aus

2.2.12 Was sind Greedy Algorithmen?

Greedy Algorithmen, sind Algorithmen, die je die vielversprechendste Teillösung wählen, bis die Gesamtlösung herauskommt.

2.2.13 Welche Beispiele kennen Sie, wo der Greedy Ansatz besonders gut, bzw. besonders schlecht ist?

Besonders gut bei MST, besonders schlecht bei TSP (Man kann eine Instanz bilden, so dass Greedy-TSP beliebig schlecht wird).

2.3 Pseudo-Polynomialzeit Algorithmen

2.3.1 Was ist ein Integer-Valued Problem?

Ein Integer-Valued Problem ist ein Problem, dessen Eingabe als Menge von Ganzzahlen gesehen werden kann. Zum Beispiel TSP, MST, VC, ...

2.3.2 Was ist ein Pseudo-Polynomialzeit Algorithmus?

Ein Algorithmus A für ein Integer-Valued Problem U heißt Pseudo-Polynomiell, wenn

- er U löst
- für alle Instanzen $x \in U$ ein Polynom p existiert, so dass $Time(x) \leq O(p(|x|, Max-Int(x)))$

2.3.3 Erklären Sie DPKP (Dynamic Programming Knapsack Algorithmus)

Der DPKP Algorithmus erhält als Eingabe eine Instanz des Knapsack Problems, also $I = (c_1, ..., c_n, w_1, ..., w_n, b)$. Er arbeitet mit Tripeln der Form: (k, w, T) mit:

- $k :=$ Summe der Kosten c_i
- $w :=$ Summe der Gewichte w_i
- $T :=$ Menge der Indizes über die summiert wird

1. Bilde $Triple(1) = \{(0, 0, \emptyset) \bigcup \{(c_1, w_1, \{1\} \mid falls\ w_1 \leq b)\}$
2. Bilde in jeweils die Menge $Triple(i+1)$ aus $Triple(i)$. Setze dazu zunächst $Set(i+1) := Triple(i)$ und überprüfe für jedes Tripel $(k, w, t) \in Triple(i)$ ob $w + w_{i+1} \leq b$ ist. Ist dem so, wird das Triple $(k + c_{i+1}, w + w_{i+1}, T \bigcup \{i+1\})$ zu $Set(i+1)$ hinzugefügt. Übernehme dann alle Triple von $Set(i+1)$ zu $Triple(i+1)$ bis auf die, die gleiche Kosten haben. Bei den Tripeln, die gleiche Kosten haben, nehme nur das Triple aus $Set(i+1)$ in $Triple(i+1)$, das das kleinste Gewicht hat.
3. Suche das Triple $Triple_{max} = (k, w, T)$ mit den maximalen Kosten und gebe $Triple_{max}$ aus

Der Algorithmus berechnet immer die optimale Lösung, da er alle Möglichkeiten berechnet.

2.3.4 Zeigen Sie: DPKP ist ein Pseudo-Polynomialzeitalgorithmus

Schritt 1 läuft in $O(1)$. Schritt 2 benötigt maximal $O(|Triple(i+1)|)$ also mit: $|Triple(i)| \leq \sum_{i=1}^{n} c_i \leq n \cdot Max - Int(I)$ und $n \leq I$ folgt, dass Step 2 in $O(|I|^2 \cdot Max-Int(I))$ liegt. Step 3, also die Bestimmung des Maximums liegt in $O(|Triple(n)|) \leq O(n \cdot Max - Int(I))$. Insgesamt gilt also $Time_A(I) \leq O(|I|^2 \cdot Max - Int(I))$ und A ist ein Pseudo-Polynomialzeitalgorithmus.

2.4 Parametrisierte Komplexität

2.4.1 Erklären Sie parametrisierte Komplexität

Ein Algorithmus A für ein Problem U ist Par-Parametrisiert, wenn

1. A löst U
2. Für jede Instanz $x \in L$ existiert ein Polynom p und eine Funktion $f : N \to N$, so dass $Time_A(x) \leq p(|x|) \cdot f(Par(x))$

2.4.2 Wie ist diese Parametrisierung Par(x) definiert?

Sei U ein Problem und L die Sprache aller Instanzen von U. Jede Funktion $Par : L \to N$ ist eine Parametrisierung für U wenn gilt:

- $Par(x)$ ist polynomzeit berechenbar
- Für unendlich viele $k \in N$ ist die Menge $Set_k(x) = \{x \in L \mid Par(x) = k\}$ unendlich

2.4.3 Welche Parametrisierten Algorithmen kennen Sie?

Zwei für das Vertex-Cover Problem, einen in $O(n \cdot k^{2k})$ und einen in $O(n \cdot 2^k)$.

2.4.4 Erklären Sie beide

2.5 Lowering Worst-Case Complexity

2.5.1 Sie haben ein Konzept kennengelernt, welches mit Lowering Worst-Case Complexity benannt wurde, was ist dies und welche Algorithmen kennen Sie?

Bei diesem Konzept nimmt man exponentielles Wachstum in Kauf, versucht jedoch Algorithmen zu finden, die ein schwach exponentielles Wachstum haben (Bsp. 1.2^n,...). Im Buch wird ein Algorithmus zu diesem Konzept vorgestellt. Durch Ihn lässt sich das 3SAT-Problem in $O(1.84^n)$ lösen.

2.5.2 Erklären sie diesen Algorithmus (3SAT in $O(1.84^n)$)

2.5.3 Schreiben Sie die Rekursionsgleichung auf und zeigen Sie, dass die Komplexität in $O(1.84^n)$ liegt

$Time_A(n, r) \leq 54r + Time_A(n-1, r-1) + Time_A(n-2, r-1) + Time_A(n-3, r-1)$
Man zeigt dann per Induktion, dass hieraus folgt: $Time_A(n, r) \leq 27r \cdot (1.84^n - 1)$.

2.6 Approximationsalgorithmen

2.6.1 Was sind Approximationsalgorithmen?

Ein Approximationsalgorithmus ist ein Algorithmus für ein Optimierungsproblem U, der $Opt_U(I)$ bis auf einen relativen Fehler $\varepsilon_A(I)$ annähert und dabei sowohl in der Zeit, als auch in der Güte abschätzbar ist.

2.6.2 Was ist der relative Fehler, was ist die Güte einer Approximation?

Der Relative Fehler von A bezüglich x und U ist definiert als: $\varepsilon_A(x) = \frac{|cost_A(x) - Opt_U(x)|}{Opt_U(x)}$
Die Approximationsgüte von A bezüglich x und U ist definiert als:
$R_A(x) = min\{\frac{cost_A(x)}{Opt_U(x)}, \frac{Opt_U(x)}{cost_A(x)}\} = 1 + \varepsilon_A(x)$

2.6.3 Welche Approximationsalgorithmen kennen Sie?

GMS 2-Approximation, SKP 2-Approximation, PTAS, FPTAS, Δ-TSP 2 Approximation, Christophides.

2.6.4 Erklären Sie 2-Approximation für GMS (incl. Güte)

2.6.5 Erklären Sie Christophides (incl. Güte)

2.6.6 Was ist ein PTAS-Algorithmus?

Ein Approximationsalgorithmus A ist ein PTAS (polynomial time approximation scheme) für U, wenn A eine Lösung für U mit einem relativen Fehler von höchstens ε liefert und dabei $Time_A(x, \varepsilon^{-1})$ in einem Polynom über $|x|$ beschränkt ist.

2.6.7 Erklären Sie den PTAS für das SKP

Eingabe: $I = \{w_1, ..., w_n, b\}$, $\varepsilon \in R^+$

1. Sortiere w_i's ($w_n \leq w_{n-1} \leq ... \leq w_1$)
2. $k = \lfloor \frac{1}{\varepsilon} \rfloor$
3. Bilde alle Mengen $S \subseteq \{1, ..., n\}$ mit $|S| \leq k$ und $\sum_{i \in S} w_i < b$ in Lexikographischer Reihenfolge.
   Erweitere alle Mengen S zu S^* mit Greedy-Ansatz uns speichere je das beste S^*.

Output: $S*$

Komplexität und Güte bitte dem Buch entnehmen.

2.6.8 Was ist ein FPTAS-Algorithmus?

Ein Approximationsalgorithmus A ist ein FPTAS (fully polynomial time approximation scheme) für U, wenn A eine Lösung für U mit einem relativen Fehler von höchstens ε liefert und dabei $Time_A(x, \varepsilon^{-1})$ in einem Polynom über $|x|$ und ε^{-1}beschränkt ist.

2.6.9 Erklären Sie den FPTAS für KP

Eingabe: $I = (c_1, c_2, ..., c_n, w_1, w_2, ..., w_n, b)$, $\varepsilon \in R^+$, $n \in N$

1. $c_{max} = max(c_1, c_2, ..., c_n)$
   $t = \lfloor log \frac{c_{max} \cdot \varepsilon}{(\varepsilon+1) \cdot n} \rfloor$
2. for $i = 1$ to n do
   $c'_i = \lfloor c_i \cdot 2^{-t} \rfloor$
3. Berechne eine Optimale Lösung T' für $I' = (c'_1, c'_2, ..., c'_n, w_1, w_2, ..., w_n, b)$ mit dem DPKP Algorithmus und gebe diese aus.

Output: T'

2.6.10 Zeigen Sie, dass dieser Algorithmus ein FPTAS ist

Komplexität:
Step 1) $\in O(n)$
Step 2) $\in O(n)$
Step 3) DPKP läuft in $O(n \cdot Opt(I'))$
$Opt(I') \leq \sum_{i=1}^{n} c'_i = \sum_{i=1}^{n} \lfloor c_i \cdot 2^{-\lfloor log \frac{c_{max} \cdot \varepsilon}{(\varepsilon+1) \cdot n} \rfloor} \rfloor \leq \sum_{i=1}^{n} (c_i \cdot 2 \cdot \frac{(\varepsilon+1) \cdot n}{c_{max} \cdot \varepsilon})$
$= 2 \cdot (1 + \varepsilon) \cdot \varepsilon^{-1} \cdot \frac{n}{c_{max}} \cdot \sum_{i=1}^{n} c_i \leq 2 \cdot (1 + \varepsilon) \cdot \varepsilon^{-1} \cdot n^2 \in O(\varepsilon^{-1} \cdot n^2)$
$\Rightarrow Step3) \in (\varepsilon^{-1} \cdot n^3)$

Also läuft der Algorithmus in $O(\varepsilon^{-1} \cdot n^3)$

Güte: siehe Buch

2.7 Heuristiken

2.7.1 Was sind Heuristiken?

Heuristiken sind randomisierte Algorithmen, die nicht gleichzeitig in der Zeitkomplexität und in der Güte beschränkbar sind.

2.7.2 Erklären Sie Simulated Annealing

Simulated Annealing ist eine Heuristik, die aus der Festkörper-Physik stammt. Hierbei wird ein Stoff erhitzt, bis er schmilzt und dann langsam abgekühlt um so eine perfekte Kristallstruktur zu erhalten. Übertragen auf die Informatik und auf das Lösen von Problemen kann die Analogie wie folgt hergestellt werden:
Wir wählen zunächst eine zufällige Mögliche Lösung $\alpha \in M(x)$, eine Starttemperatur T und eine Temperatur-Reduktions-Funktion. Nun wählt man zufällig ein $\beta \in Neigh(\alpha)$ und ersetzt α durch diese wenn sie besser ist, oder wenn sie schlechter ist, ggf. (durch Zufall in Abhängigkeit von T). Dies wird so lange wiederholt, bis $T = 0$ ist. Je höher T ist, desto wahrscheinlicher wählen wir eine schlechtere Lösung. Je niedriger T wird, desto stärker nähert sich dieses verfahren der lokalen Suche an.

2.7.3 Erklären Sie genetische Algorithmen

Kapitel 3

Kryptographie

3.1 Symmetrische Verschlüsselung

3.1.1 Was ist der Unterschied zwischen symmetrischer- und public-key Verschlüsselung?

Bei der symmetrischen-Verschlüsselung besitzen Sender und Empfänger den selben Schlüssel. Dieser ist geheim zu halten, sowie initial geheim zu übertragen. Kennt jemand diesen Schlüssel, so kann er alle Kryptogramme entschlüsseln. Bei der Public-Key Kryptographie hingegen gibt es je einen Public- und einen Secret-Key. Der Public-Key wird zur Verschlüsselung genutzt, der Secret Key zur Entschlüsselung. Es muss initial kein Gehimnis ausgetauscht werden.

3.1.2 Erklären Sie den DES-Algorithmus

Der DES-Algorithmus ist ein symmetrisches Blockchiffre-Verfahren. Das bedeutet, die Nachricht wird Blockweise verschlüsselt. DES verwendet dazu folgende Funktion: $DES_k : \{0,1\}^{64} X \times \{0,1\}^{56} \rightarrow \{0,1\}^{64}$

1. Generiere aus dem Schlüssel k mit $|k| = 56\ Bits$ 16 Rundenschlüssel k_i mit je 48 Bits durch Permutationen und Shifts.
2. Permutiere den Nachrichtenblock m initial (IP)
3. Teile den Nachrichtenblock m in $L_0 R_0$ mit $|L_0| = |R_0| = 32\ Bits$
4. $L_1 = R_0$ und $R_1 = L_0 \oplus f(R_0, k_1)$
   .....
   .....
   $L_{16} = R_{15}$ und $R_{16} = L_{15} \oplus f(R_{15}, k_{16})$
5. Permutiere den $L_{16} R_{16}$ (IP^{-1}) und erhalte c

Dabei ist f die eigentliche Verschlüsselungsfunktion mit $f : \{0,1\}^{32} X \times \{0,1\}^{48} \rightarrow \{0,1\}^{32}$

1. Erweitere R_i auf 48-Bits
2. $R'_i = R_i \oplus k_i$
3. Teile R'_i in 8 Blocks a 6 Bit und Substituiere Sie durch die S-Boxen mit je 4 Bit.
4. Permutiere die Blöcke (P) und erhalte R_{i+1}

3.1.3 Können Sie die Eindeutigkeit von DES beweisen?

Es gilt: $DES_k = IP^{-1} \circ \phi_{16} \circ \mu \circ ... \circ \mu \circ \phi_1 \circ IP$
Aus $\mu(x, y) = (y, x)$ folgt unmittelbar: $\mu = \mu^{-1}$
Da $\phi \circ \phi(x, y) = \phi(x \oplus f_i(y), y) = (x \oplus f_i(y) \oplus f_i(y), y) = (x, y)$ folgt $\phi = \phi^{-1}$ und somit:
$DES_{k_{16}...k_1} \circ DES_{k_1...k_{16}}(x) = x$

3.1.4 Schreiben Sie die Cipher-Feedback Mode auf

Nachricht $m = m_1...m_l$ mit $|m_i| = r$ und Blocklänge n

1. Wähle $x_0 \in \{0, 1\}^n$ zufällig
2. $c_i = m_i \oplus msb_r(f(x_i))$
3. $x_{i+1} = lsb_{n-r}(x_i)||c_i$

3.2 Public-Key Kryptographie

3.2.1 Welche mathematischen Probleme eignen sich für die Public-Key Verschlüsselung?

Eigentlich jede One-Way fuunktion mit Trapdoor. Insbesondere: Modulare Exponenten (RSA), diskreter Logarithmus (El Gamal) und Modulares Quadrieren (Rabin).

3.2.2 Erklären Sie RSA

RSA ist ein Public-Key Verfahren, was auf dem Problem der modularen Exponenten basiert. Es wurde von Rivest, Shamir und Adleman entwickelt.

Schlüsselgenerierung:

1. Wähle zwei Primzahlen p, q mit $n = p \cdot q$
2. Wähle ein e, welches Teilerfremd zu $\varphi(n)$ ist
3. Bestimme ein d mit $e \cdot d \equiv 1 \; mod \; \varphi(n)$
4. Public-Key: (n, e) Private-Key: (n, d)

Ver- bzw. Entschlüsselung:

1. $E : m \rightarrow m^e$
2. $D : c \rightarrow c^d = (m^e)^d = m$

3.2.3 Warum muss e teilerfremd zu $\varphi(n)$ sein?

Das liegt darin begründet, dass sonst kein Inverses d mit $e \cdot d \equiv 1 \; mod \; \varphi(n)$ existiert.

3.2.4 Können Sie beweisen, dass die Entschlüsselung eindeutig ist?

Um zu zeigen, dass RSA eindeutig ist, muss man zeigen, dass $w^{ed} \equiv w \ mod \ n$ gilt.

Fall1: (weder p, noch q teilen w)
$w^{\varphi(n)} \equiv 1 \ mod \ n$ (Euler Theorem)
$w^{j \cdot \varphi(n)} \equiv 1 \ mod \ n$
$w^{ed-1} \equiv 1 \ mod \ n \ (ed \equiv 1 \ mod \ \varphi(n) \Rightarrow \varphi(n) \ vielfaches \ von \ ed-1)$
$w^{ed} \equiv 0 \ mod \ n$
$w^{ed} \equiv w \ mod \ n$

Fall 2 und Fall3, siehe Buch!

3.2.5 Erklären Sie, wie die digitale Signatur mit RSA funktioniert

3.2.6 Erklären Sie das OAEP

3.2.7 Erklären Sie Merkle's Meta Method

3.2.8 Erklären Sie El-Gamal

El Gamal ist ein Public-Key Verschlüsselungsverfahren, dass auf dem mathematischen Problem des diskreten Logaritmus beruht.

Schlüsselgenerierung:

1. Wähle eine Primzahl p mit $p-1$ großem Primfaktor
2. Wähle einen Generator $g \in Z_p^*$
3. Wähle ein $x \in \{0, ..., p-2\}$ zufällig
4. Berechne $y = g^x$
5. Public-Key: (p, g, y) Private-Key: (p, g, x)

Verschlüsselung:

1. Klartext: m, Cryptotext: c
2. Wähle $k \in \{0, ..., p-2\}$ zufällig
3. $c = (g^k, y^k \cdot m)$

Entschlüsselung:

1. $m = y^k \cdot m \cdot (g^k)^{-x} = (g^x)^k \cdot m \cdot g^{-xk} = g^{xk} \cdot g^{-xk} \cdot m = g^{xk-xk} \cdot m = m$

3.2.9 Erklären Sie die Signatur mit El-Gamal

3.2.10 Erklären Sie Rabin

Rabin ist ein Public-Key Verfahren, dass auf dem mathematischen Problem des modularen quadrierens basiert. Im Gegensatz zu El-Gamal und RSA ist beweisbar so schwer wie die Faktorisierung großer Zahlen.

Schlüsselgenerierung:

1. Wähle zwei große Primzahlen $p,\ q$ mit $n = pq$ (Gut für die Entschlüsselung: $p, q \equiv 3\ mod\ 4$)
2. Public-Key: (n) Private-Key: (p, q)

Verschlüsselung:

1. $E : x \to x^2$

Enschlüsselung:

1. Zur Bestimmung von $x \equiv \sqrt{c}\ mod\ n$ berechne:
2. $a \equiv c\ mod\ p$
   $b \equiv c\ mod\ q$
3. Ziehe dann die Wurzeln von a, b in Z_p, Z_q
   $x \equiv \sqrt{a}\ mod\ p$
   $x \equiv \sqrt{b}\ mod\ q$
4. Erhalte so bis zu 4 Lösungen, und wähle (durch Zusatzinformation, oder durch raten) die richtige Lösung. Mit dem Chinesischen Restsatz folgt dann $x \equiv \sqrt{c}\ mod\ n$

3.2.11 Erklären Sie den Chinesischen Restsatz

Wenn ein System von Kongruenzen der Form $x \equiv a_1\ (mod\ m_1), x \equiv a_2\ (mod\ m_2), ...$ mit $m_i \in N$ paarweise relativ prim und $a_i\ inZ$ gegeben ist, dann gibt es ein eindeutig bestimmtes x im Bereich $0 \leq x \leq m_1 \cdot m_2 \cdot ... \cdot m_n$.

3.2.12 Erklären Sie die Signatur mit Rabin

3.3 Protokolle

3.3.1 Erklären Sie Strong-Three-Way

3.3.2 Erklären Sie Station-to-Station

Das Station-to-Station Protokoll ist eine Art Diffie-Hellmann Key Agreement Protokoll mit einer Mutal Authentification. Zunächst wird eine Primzahl p mit $p - 1$ großem Primfaktor gewählt. Dann sucht man einen Generator $g \in Z_p^{\cdot}$.

1. Alice wählt ein $a \in \{0, ..., p - 2\}$ und sendet $c = g^a$ an Bob
2. Bob wählt ein $b \in \{0, ..., p - 2\}$ berechnet $k = (g^a)^b$, sowie $s = Sign_B(g^a||g^b)$ und sendet $(g^b, E_k(s))$ an Alice
3. Alice Überprüft s und berechnet $s' = Sign_A(g^b||g^a)$. Dann sendet Alice $E_k(s')$ an Bob.

3.3.3 Worauf beruht die Sicherheit des Station-2-Station Protokolls?

Die Sicherheit des Station-2-Station Protokolls beruht auf der Diffie-Hellmann Assumption, sie besagt, dass g^{ab} unmöglich aus der Kenntnis von g^a und g^b berechnet werden kann.

3.3.4 Welche Bedingungen gelten für ein Interactive Proof-System?

Ein Interactive Proof System ist durch folgende Eigenschaften gekennzeichnet:

1. Vollständigkeit: Wenn Peggy das Geheimnis kennt, muss Vic akzeptieren.
2. Verläßlichkeit: Überzeugt Peggy Vic mit hinreichender Wahrscheinlichkeit, so muss Peggy das Geheimnis kennen.

3.3.5 Was bedeutet, wenn ein Beweissystem Zero-Knowledge ist?

Ein interaktives Beweissystem zwischen Prover und Verifier (P, V) ist Zero-Knowledge, wenn ein probabilistischer Simulator existiert $S(V^{\cdot}, x)$, der für einen beliebigen Verifier $V^{\cdot}$ ein akzeptierendes Transkript $tr_{P,V^{\cdot}}(x)$ in erwarteter Polynomzeit berechnet. Dieses Transkript darf in seiner Verteilung nicht von einem normalen zu unterscheiden sein.

3.3.6 Welche Zero-Knowledge Protokolle kennen Sie?

Simplified Fiat-Shamir

3.3.7 Erklären Sie Simplified Fiat-Shamir

- $n = pq$, $QR_n :=$ Menge der Quadrate in Z_n
- $x \in QR_n$ und $y^2 = x$. y ist Peggy's Geheimnis.

Das Protokoll durchläuft folgende Schritte:

1. Peggy wählt eine Zahl $r \in Z_p^{\cdot}$ zufällig und sendet $a = r^2$ an Vic
2. Vic wählt ein $e \in \{0, 1\}$ zufällig und sendet dieses an Peggy.
3. Peggy berechnet $b = r \cdot y^e$ und sendet b an Vic
4. Vic überprüft, ob $b^2 = a \cdot x^e = r^2 \cdot (y^2)^e = r^2 \cdot (y^e)^2 = (r \cdot y^e)^2 = (b)^2$

3.3.8 Warum ist dieses Protokoll Zero-Knowledge?

Dieses Protokoll ist Zero-Knowledge, da ein probabilistischer Simulator wie folgt zu erstellen ist:
Der Simulator wählt zunächst ein $\widetilde{e}$ und schickt $a = r^2 \cdot x^{\widetilde{e}}$ an Vic. Vic, egal ob honest oder nicht sendet ein e an den Simulator. Wenn $e = \widetilde{e}$ sendet der Simulator $b = r$ notiert der Simulator den Schritt (a,e,b). Das Transkript kann dabei in Polynomzeit erstellt werden, weil der Simulator mit der Wahrscheinlichkeit $p = 0,5$ das richtige e rät.

3.4 Commitment Schemes

3.4.1 Welche Bedingungen gelten für Commitment Schemes?

1. Hiding Property: B darf aus dem verschlüsselten Commitment nichts erfahren dürfen
2. Binding Property: A muss an sein Commitment gebunden sein
3. Viability: Wenn A und B sich an das Protokoll halten, wird B immer das Geheimnis erhalten

3.4.2 Erklären Sie das Commitment Scheme mit der Quadratischen Restklasse

1. Alice wählt zwei Primzahlen p, q mit $n = pq$ und $v \in QNR_n^{+1}$
2. (Commit) Alice legt ihr Commitment fest $b \in \{0, 1\}$, wählt zufällig $r \in Z_n$ und berechntet $c = r^2 v^b$ Dann sendet Alice (n, c, v) an Bob
3. (Reveal) Alice sendet (p, q, r, b) an Bob und Bob testet, ob p, q Primzahlen, ob $n = pq$, ob $r \in Z_n$, ob $v \in QR_n$ und ob $c = r^2 v^b$.

3.4.3 Erklären Sie das Commitment Scheme mit dem diskreten Logarithmus

1. Bob wählt p, q mit $q \backslash p - 1$ und zufällig $g, v \in G$ (mit G Subgruppe mit Ordnung q in Z_p) und sendet (p, q, g, v) an Alice
2. (Commit) Alice prüft (p, q, g, v) und wählt Commitment $m \in \{0, ..., q - 1\}$ und zufälliges $r \in \{0, ..., q - 1\}$. Alice berechnet $c = g^r v^m$ und sendet c an Bob.
3. (Reveal) Alice sendet m, r

3.4.4 Was sind homomorphe Commitments?

Commitments für die gilt: $Com(r_1, m_1) \cdot Com(r_2, m_2) = Com(r_1 + r_2, m_1 + m_2)$
Homomorphe Commitments werden für elektronische Wahlen verwendet

3.5 Elektronische Wahlen

3.5.1 Welche Anforderungen werden an elektronische Wahlen gestellt?

1. Universal Verifiability: Ergebnis der Wahl kann von jedem überprüft werden
2. Privacy: Stimmabgabe ist geheim
3. Robustness: System funktioniert auch dann, wenn mehrere Teilnehmer betrügen

3.5.2 Erklären Sie kurz den Aufbau einer elektronischen Wahl

- Nur Yes/No Votes
- Wähler wählt und verschlüsselt seine Stimme mit homomorphem Verschlüsselungsalgorithmus
- Verschlüsselte Stimmen werden gesammelt (Bulletin Board)
- Autorität kann mit geheimem Schlüssel das Wahlergebnis berechnen
- Damit betrügen schwieriger wird, teilt man den geheimen Schlüssel auf n Autoritäten auf, von denen je t-Stück den geheimen Schlüssel berechnen können ((t,n)-threshold Scheme)
- Verschlüsselung: El Gamal
- Beim Entschlüsseln wird nie eine einzelne Stimme berechnet
- Mit Proof of knowledge kann getestet werden, ob betrogen wurde

3.6 Digital Cash

3.6.1 Welche Anforderungen werden an Digital-Cash gestellt

Wichtig beim Digital Cash sind drei Dinge:

- Es kann nur dann ein Coin eingelöst werden, wenn es zuvor abgehoben wurde. Ein Coin darf keine zwei mal eingelöst werden, ohne das dies bemerkt wird
- Der Kunde kann mit einem Coin, dass er zuvor abgehoben hat anonym im Laden bezahlen
- Der Verkäufer muss sichergehen können, dass das Coin des Kunden später von der Bank akzeptiert wird

3.6.2 Erklären Sie kurz die Funktionsweise von Digital-Cash

Konto eröffnen:
Kunde beweist seine Identität der Bank

Geld abheben:
Kundeidentifiziert sich bei der Bank (Interactive Proof). Daraufhin signiert die Bank ein Coin durch die sogenannte Blind Signature und übermittelt die 'Blindmacher' an das Trusted Center. Im Anschluss speichert die Bank die Identität des Benutzers, die jedoch nur mit Hilfe des Trusted Centers revealed werden kann. Durch Proof's of Knowledge wird sichergestellt, dass jeder das sendet, was er soll.

Bezahlen:
Kunde sendet Coin=(m, δ) an Shop. Dieser verifiziert die Signatur der Bank und sendet das Coin an die Bank. Die Bank testet, ob das Coin bereits verwendet wurde und ob δ gültig ist und akzeptiert.

Owner Tracing:
Trusted Center kennt (u, v, w) und kann $(\overline{a}, \overline{b}, \overline{c})$ mit blindem (c, b) in Verbindung bringen.

Kapitel 4

Automatentheorie

4.1 Automaten und Logik-Formeln

4.1.1 Definieren Sie einen NEA

Ein NEA ist ein 5-Tupel, $(Q, \Sigma, q_o, \Delta, F)$ mit:

- Q := Endliche Zustandsmenge
- Σ := Alphabet
- q_0 := Startzustand
- F := Menge der Endzustände
- Δ := Transitionsrelation mit $\Delta \in (QX \times \Sigma X \times Q)$

4.1.2 Erklären Sie die MSO-Logik

MSO-Logik steht für Monadic-Second-Order-Logic. Sie besteht aus:

- Variablen über Buchstabenpositionen: $x, y, z, ...$ und Mengenvariablen $X, Y, Z, ...$
- Aus Konstanten: min,max
- Aus atomaren Formeln: $x = y$; $x < y$; $P_a(x)$; $X(x)$; $S(x, y)$
- Junktoren: $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$
- Quantoren: $\exists, \forall$

MSO-Formeln werden in sogenannten Wortmodellen interpretiert.

4.1.3 Welche Sprachen lassen sich durch MSO-Formeln beschreiben?

Nach Satz von Büchli, Elgot und Trachtenbrot gilt: Eine Sprache ist MSO-Definierbar genau dann wenn sie regulär ist. Also definieren die MSO-Formeln genau die regulären Sprachen.

4.1.4 Wie kann man aus einem Automat eine Formel konstruieren?

Die Konstruktion sieht wie folgt aus: Zu einem NEA $A = (\{1, ..., m\}, \Sigma, 1, \Delta, F)$ konstruiert man eine MSO Formel $\phi =$

- $\exists X_1 ... \exists X_m [\forall x(X_1(x) \vee ... \vee X_m(x)) \wedge \bigwedge_{i \neq j} \neg \exists x(X_i(x) \wedge X_j(x))]$
- $\wedge X_1(min)$
- $\wedge \forall x \forall y(S(x, y) \rightarrow \bigvee_{(i,a,j) \in \Delta}(X_i(x) \wedge P_a(x) \wedge X_j(y)))$
- $\wedge \bigvee_{(i,a,j) \in \Delta, j \in F}(X_i(max) \wedge P_a(max))$

Dann gilt: A akzeptiert w gdw. $\underline{w} \models \phi$

4.1.5 Erklären Sie kurz, wie die Übersetzung von Formeln in Automaten geht

Um eine MSO-Formel in einen Automaten zu überführen muss man zunächst die Formel in die Äquivalente MSO_0 Form transformieren. Diese enthält nur noch Monadische Variablen 2. Stufe. Dann baut man für die atomaren Formeln Automaten und verkettet diese bezüglich der Operatoren und Junktoren (Induktiver Vorgehen über den Formelaufbau).

4.1.6 Was sind sternfreie Ausdrücke

Sternfreie Ausdrücke sind verallgemeinerte reguläre Ausdrücke ohne den Sternoperator. Verallgemeinerte reguläre Ausdrücke sind reguläre Ausdrücke mit den zusätzlichen Operationen: $\sim, \cap$.

4.1.7 Erklären Sie die LTL-Logik

Die LTL Logik ist eine Temporale Logik, die wie folgt aufgebaut ist:

- Aus temporalen Variablen: $p_1, p_2, ..., p_n$
- Aus Junktoren $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$
- Und aus temporalen Operatoren: G (Always), F (Eventually), U (Until) und X (Nexttime)

4.1.8 Was können Sie über den Zusammenhang zwischen LTL Logik und sternfreiheit sagen?

Es gelten die Implikationen: LTL-Definierbar $\Leftrightarrow$ FO-Definierbar $\Leftrightarrow$ sternfrei. Die Implikationen können je per Induktion bewiesen werden.

4.1.9 Wann ist eine Sprache nichtzählend?

Eine Sprache heißt nichtzählend, wenn ein n_0 existiert, so dass für alle $n \leq n_0,\ u, w \in \Sigma^{\cdot} und\ v \in \Sigma^+$ gilt:
$uv^n w \in L \Leftrightarrow uv^{n+1} w \in L$

4.2 Äquivalenz und Minimierung

4.2.1 Was ist ein NEA-Homomorphismus?

Ein NEA Homomorphismus zwischen zwei NEA's A,B ist eine Funktion: $h : Q^A \rightarrow Q^B$ mit:

- $h(q_0^A) = q_0^B$
- $(p, a, q) \in \Delta^A \Rightarrow (h(p), a, h(q)) \in \Delta^B$
- $q \in F^A \Rightarrow h(q) \in F^B$

Dann gilt: $L(A) \subseteq L(B)$

4.2.2 Was ist die kanonische Zustandsäquivalenz?

Sei $A = (Q, \Sigma, q_0, \delta, F)$ ein DEA. Dann ist die Kanonische Zustandsäquivalenz $\sim_A$ zwischen p und q wie folgt definiert:
$p \sim_A q \Leftrightarrow L(A_p) = L(A_q)$

4.2.3 Auf welcher Kongruenz basiert die DEA Minimierung?

Auf der Nerode Konquenz. Die Nerode Kongruenz ($\sim_L$) ist wie folgt definiert: $u \sim_L v \Leftrightarrow \forall w \in \Sigma^{\cdot} gilt : uw \in L \Leftrightarrow vw \in L$

4.2.4 Was ist der kanonische DEA?

Der Kanonische DEA für L ist ein DEA, der die Sprache L erkennt und unter allen L-erkennenden DEA's kleinstmögliche Zustandszahl hat. $A_L = (Q, \Sigma, q_0, \delta, F)$ mit:

- Q = Menge der $\sim_L$ Klassen
- $q_0 = [\varepsilon]$
- $F = \{[p] | p \in L\}$
- $\delta([u], a) = [ua]$

4.2.5 Erklären Sie die DEA-Minimierung

Die Idee bei der DEA Minimierung ist es Wörter zu finden, die nicht-Äquivalente Zustände trennen. Dabei geht man über die Länge der Wörter und beginnt mit dem leeren Wort. Zunächst teilt man die Zustandsmenge in zwei Blöcke, nämlich F und $Q \backslash F$, da diese durch das leere Wort getrennt werden un daher nicht äquivalent sind. Diese Blockeinteilung wird dann schrittweise durch den sogenannten Blockverfeinerungs-Algorithmus verfeinert. Dieser spaltet in einem Schritt einen Block B_i bezüglich B_j und a auf wenn gilt: $p, q \in B_i$ und $\delta(p, a) \in B_j$ aber $\delta(q, a) \notin B_j$. Der Algorithmus verfeinert die Blöcke so lange, bis keine Aufteilung mehr möglich ist.

4.2.6 Wie ist die Komplexität des Algorithmus?

4.2.7 Welche Ansätze gibt es zur NEA-Minimierung?

Da gibt es zum einen die Suboptimale Minimierung über die Bisimulations-Äquivalenz und die suche des minimalen NEA im sogenannten Universal-NEA

4.2.8 Erklären Sie den Algorithmus zu NEA-Minimierung

Die NEA-Minimierung kann man nicht wie beim DEA über die Sprachäquivalenz machen, da die Berechnung der nicht-äquivalenten Zustände für NEA's schwierig ist. Anstatt dessen verwendet man hier die sogenannte Bisimulations-Äquivalenz, die feiner ist als die Sprachäquivalenz. Der Algorithmus läuft dabei wie folgt ab:

1. Zunächst wird der NEA in einen äquivalenten NEA transformiert, der keine ausgehenden Kanten in seinen Endzuständen besitzt.
2. Dann markiert man alle Zustandspaare (p, q) wo entweder $p \in F$, $q \notin F$ oder $q \in F$, $p \notin F$, da Endzustände und nicht-Endzustände auch nicht bisimilar sind.
3. in jedem Schritt wird dann ein Paar (p, q) markiert, wenn es von p eine Transition (p, a, p') und von q eine Transition (q, a, q') gibt, so dass (p', q') nicht bisimilar sind.

4.2.9 Wie ist die Laufzeit des Algorithmus?

Der Algorithmus Terminiert spätestens nach $|P| \cdot |Q|$ Schritten. In jedem Schritt benötigt er höchstens $|P| \cdot |\Delta_A| + |Q| \cdot |\Delta_B|$ Schritte. Daraus ergibt sich eine Laufzeit $Time_A(x) \leq O(|Q|^3 \cdot |\Delta|)$.

4.2.10 Was ist der Universal-NEA. Erklären Sie die Konstruktion

4.3 Baumautomaten

4.3.1 Was ist ein Rangalphabet?

Ein Rangalphabet $\Sigma = \Sigma_0 \cup ... \cup \Sigma_n$ ist eine nicht leere Menge von Symbolen, denen ein Rang zugeordnet ist. Dieser Rang bezeichnet die Anzahl der Nachfolger des Symbols. Beispiel: $\Sigma_0 = \{0, 1\}$; $\Sigma_1 = \{\neg\}$; $\Sigma_2 = \{\vee, \wedge\}$

4.3.2 Wozu können Baumautomaten genutzt werden? Geben Sie ein Beispiel

Baumautamten sind Automaten, welche auf Bäumen operieren. Sie können beispielsweise Formeln, wie die zu 1 auswertbaren aussagelogischen Ausdrücke, erkennen.

4.3.3 Definieren Sie einen DBA

Ein DBA ist ein 4-Tupel: (Q, S, Δ, F) mit:

- Q := endliche Zustandsmenge
- $\Sigma := \bigcup_{i=0}^{k} \Sigma_i$, k := maximale Anzahl der Nachfolger
- Transitionsfunktion S: $\bigcup_{i=0}^{k} (Q^i \times \Sigma_i) \rightarrow Q$
- F := Menge der Endzustände

4.3.4 Was ist ein spezieller Baum?

Ein Spezieller Baum ist ein Baum, dessen Rangalphabet um das Symbol $\Sigma_0 = \{c\}$ erweitert wird. Dieses Symbol kommt nur genau einmal im Baum vor und kann dann für die Konkatenation zweier Bäume durch den anderen Baum ersetzt werden. Spezielle Bäume werden beim Pumping Lemma, sowie zur Definition des Kleene-Sterns über Bäumen verwendet.

4.3.5 Erkenne NBA's und DBA's die selben Sprachen?

Ja, da erstens jeder DBA auch als NBA gesehen werden kann. Und zweitens aus jedem NBA, durch die von endlichen Automaten über Sprachen bekannte Potentzmengenkonstruktion, ein äquivalenter DBA erstellt werden kann.

4.3.6 Erklären Sie die Funktionsweise der Top-Down-Baumautomaten

Top-Down Baumautomaten beginnen anders als die Bottom-Up Automaten an der Wurzel und werten den Baum von dort aus nach unten hin aus.

4.3.7 Erkennen ↓DEA's und ↓NEA's die selben Sprachen?

Nein, die durch die ↓DEA's definierten Sprachen bilden eine echte untermenge der ↓NEA definierten Sprachen. Dies kann durch das Beispiel der Zweiermenge: $f(a, b)$ und $f(b, a)$ gezeigt werden.

4.4 Kellerautomaten

4.4.1 Was ist ein PDA

Ein PDA ist definiert als 6-Tupel: $A = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$ mit:

- Q := Endlicher Zustandsmenge
- Σ := Eingabealphabet
- Γ := Arbeitsalphabet
- q_0 := Anfangszustand
- Z_0 := Kellerstartsymbol
- Δ := Transitionsrelation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma^{\cdot} \times Q)$

4.4.2 Was ist eine Konfiguration / erweiterte Konfiguration eines PDA's?

Eine Konfiguration ist ein Tupel (Zustand, Kellerinhalt) eine erweiterte Konfiguration ist ein Tripel (Zustand, Kellerinhalt, Restinput).

4.4.3 Was ist ein Konfigurationsgraph?

4.4.4 Welche Sprachen erkennen die PDA's?

Ein Sprache ist durch einen Pushdown Automaten erkennbar gdw. sie kontextfrei ist.

4.4.5 Welche Probleme sind über PDA's entscheidbar, welche nicht?

Entscheidbar sind das Leerheitsproblem, sowie das Problem, ob ein Wort erkennbar ist. Unentscheidbar sind Inklusion, bzw. Äquivalenz.

4.4.6 Was ist ein Pusdownsystem?

Ein Pushdown System ist ein Pushdown Automat bei dem die Eingabe weggelassen wird. Formal ist er definiert als Tripel: $P = (P, \Gamma, \Delta)$ mit:

- P := Endliche Zustandsmenge
- Γ := Kelleralphabet
- Δ := Transitionsrelation mit $\Delta \subseteq P \times \Gamma \times \Gamma^{\cdot} \times P$

4.4.7 Ist das Erreichbarkeitsproblem auf Pushdownsystemen entscheidbar?

Ja und zwar kann man die Menge $post^{\cdot}(L)$ zu einer Menge von Konfigurationen L durch den 2. Saturierungsalgorithmus bestimmen und dann prüfen, ob die Konfiguration, welche man erreichen möchte in der regulären Menge $post^{\cdot}(L)$ enthalten ist.

4.4.8 Ist das Erreichbarkeitsproblem auch für n-Pusdownsysteme entscheidbar?

Nein.

4.5 Kommunizierende Systeme

4.5.1 Was ist ein CFSM

Ein CFSM ist ein FIFO-Kommunizierender Automat. Formal ist er definert als: $A = (CN, \Gamma, A_1, ..., A_n)$ mit:

- $CN \subseteq CN_n := (\{1, .., n\} \times \{1, ..., n\} \backslash \{(1,1), ..., (n,n)\})$ Menge der Nachrichtenkanäle
- Γ := Nachrichtenalphabet
- $A_1, ..., A_n$ NEA's mit folgenden Transitionen:
  - $(p, a, q) \in Q_i \times \Sigma_i \times Q_i$
  - $(p, m!, j, q)$ Sende m an j und gehe von p nach q
  - $(p, m?, j, q)$ Empfange m aus j und gehe von p nach q

4.5.2 Was ist ein globaler Zustand eines CFSM?

Ein globaler Zustand ist die Menge der Zustände der NEA's $A_1, ..., A_n$ sowie der Inhalt der Nachrichtenkanäle.

4.5.3 Welche Probleme kennen Sie auf CFSM's die nicht entscheidbar sind?

Zum einen das Erreichbarkeitsproblem, zum anderen das Beschränktheitsproblem.

4.6 Petrinetze

4.6.1 Wie ist ein Petrinetz definiert?

Ein Petrinetz ist ein Tripel $P = (P, T, F)$ mit:

- P := Stellen
- T := Transitionen
- F := Flußrelation mit $F \subseteq (P \times T) \cup (T \times P)$

Die Markierung eines Petrinetzes P ist eine Funktion $m : P \to N$, die jeder Stelle eine Markenzahl zuordnet.

4.6.2 Wie sind Vor- bzw. Nachbereich eines Petrinetzes definiert?

Der Vorbereich einer Transition $t \in T$ ist definiert als $\bullet t = \{p \in P \mid (p, t) \in F\}$
Der Nachbereich einer Transition $t \in T$ ist definiert als $t\bullet = \{p \in P \mid (t, p) \in F\}$

4.6.3 In der Vorlesung wurden auch Petrinetze als Sprachakzeptoren vorgestellt, welche Sprachen werden von solchen Netzen akzeptiert?

Die Petrinetze akzeptieren alle regulären, einige Kontextfreie und einige kontextsensitive Sprachen.

4.6.4 Wann ist ein Petrinetz strikt konservativ, wann ist es konservativ?

Ein Petrinetz $N = (P, T, F)$ mit Anfangsmarkierung m^- heißt strikt konservativ, falls für alle von m^- erreichbaren Markierungen m gilt: $m_1^- + ...m_n^- = m_1 + ... + m_n$

Ein Petrinetz $N = (P, T, F)$ mit Anfangsmarkierung m^- heißt konservativ, falls für alle von m^- erreichbaren Markierungen m ein Gewichtsvektor $w = (w_1, ..., w_n)$ existiert, so dass gilt: $m^- \cdot w = m \cdot w$

4.6.5 Wie testet man ein Petrinetz auf konservativität?

Ein Petrinetz N mit Transitionsmatrix D ist konservativ gdw. $D \cdot w^T = 0$ hat eine Lösung durch einen Gewichtsvektor $w \in N_+^{|T|}$

4.6.6 Welche Probleme sind auf Petri-Netzen entscheidbar? Wie?

Auf Petrinetzen ist das Erreichbarkeits- und das Beschränktheitsproblem entscheidbar. Um die Erreichbarkeit zu testen, erstellt man den Erreichbarkeitsbaum, für die Beschränktheit erstellt man den Karp-Miller-Baum.

4.6.7 Wie konsturiert man den Karp-Miller-Baum?

Eingabe: Petrinetz $P = (P, T, F)$, Markierung m^-

- m^- := unmarkiertes Blatt

- WHILE es gibt unmarkiertes Blatt
  - wähle solches Blatt m
  - für jede Markierung m' mit $m \triangleright m'$
    - füge m' als Sohn von m hinzu
    - falls auf dem Pfad zurück zur Wurzel ein m^0 existiert mit $m^0 < m'$ schreibe ∞ für jede Komponente $m'_i > m^0$
    - falls m' tritt andernorts auf, markiere m'
    - falls m' ist Deadlock, markiere m'

Output: Baum $t(N, m^-)$

4.6.8 Welche nicht reguläre Sprache kennen Sie, die von einem Petrinetz erkannt wird?

Beispielsweise: $L = \{w \in L \mid w = a^n b^n\}$, $L = \{w \in L \mid w = a^n c b^n\}$, $L = \{w \in L \mid w = a^n b^n c^n\}$, ...

4.6.9 Zeichnen Sie das Petrinetz für $a^n b^n c^n$