Theoretische Informatik

Tony Khoa Nam Huynh

21. Oktober 2022

Inhaltsverzeichnis

1 Einführung

Sie Studieren Informatik an einer HAW, und jetzt kommt die Theoretische Informatik. Warum?!

1.1 Unterteilung der Informatik

Die zentrale Fragestellung ist, was ist (wenn überhaupt) mit welchen Aufwand berechenbar? Die THI teilt alle Probleme in sog. Sprachklassen ein, bei denen sich der Lösungsaufwand unterscheidet auf Zeit, Speicherplatz, Zugriffsmöglichkeit (Array, Stake)

1.2 Warum braucht man THI?

Szenario: Ihr Chef beauftragt Sie ein Problem zu lösen, welches wenn es nicht innerhalb von 4 Wochen gelöst die Pleite der Firma bedeutet.

1) Problem wurde nach 2 Wochen gelöst
2) Es findet sich keine optimale und schnelle Lösung
   2a) 'Chef, ich bin zu doof'
   2b) "Chef, das Problem ist nicht unlösbar, aber es gibt Tausende ähnliche Probleme, die auch alle nicht gelöst sind"
   2c) "Chef, es ist unlösbar"

1.3 Praktische Anwendung der THI

1) Suchen und Ersetzen in Texten:

   Lösche Leerzeichen aus einem Text, ersetze Punkte am Zeilenende durch Semikolon

2) Einlesen von CSV-Dateien

   - Pro Tabellenzeile ein Textzeile
   - Pro Felder getrennt durch ','
   - Text in ""verpackt
   - Innerhalb von ""wird als ""geschrieben (immer ungerade Anzahl)
   - Auswerten von arithmetischen Ausdrücken

1.4 Alphabet, Grammatiken, Sprachen

In der THI werden Problem als Entscheidungsfragen formuliert. Man fragt nicht 'Wie weit ist es von Stuttgart nach München', sondern 'Gibt es einen Weg von Stuttgart nach München, der höchstes 220 km lang ist.'

Anstelle offene Fragen 'Wie lang damit die Auslieferung dieser 5 Punkte benötigt' die geschlossene Frage 'lassen sich diese 5 Punkte in 2:15 zustellen?'

Die Eingaben müssen in einem vordefinierten Format (unter anderem mit definierten Symbolen) verfügbar sein.

Diese Symbolmenge heißt Alphabet:

Definition 1.1

Ein Alphabet $\sum = \{\sum_1, \ldots, \sum_n\}$ ist eine endliche Menge von Symbolen.

Beispiel 1.1

Beispiel 1.1: {'A', ..., 'Z', 'a', ..., 'z'} ist das latinische Alphabet, {'public', 'private', 'protected', 'class', 'abstract', ..., 'while'} ist das Alphabet der reservierten Java-Wörter.

Die Menge der Java-Bezeichner oder -Zahlen bildet kein Alphabet, weil sie nicht endlich ist.

Definition 1.2

Das Alphabet $\sum$ ist $\sum^+$ die Menge aller nicht-leeren Folgen von Elementen aus $\sum, \sum^* = \sum^+ \cup \{\varepsilon\}$. Man nennt $\sum_0^*$ auch das Wort-Monoid (Ein *Monoid* ist eine algebraische Struktur, bestehend aus einer Menge U, einer assoziativen binären Verknüpfung und einem neutralen Element, ein typisches Beispiel ist $(\mathbf{N_0}, +)$ oder $(\mathbf{N}, ^*)$).

Definition 1.3

Eine *Sprache* $\mathcal{L}$ über dem Alphabet $\sum$ ist eine beliebige Teilmenge $\sum^*$. Die *charakteristische Funktion* $\varkappa_{\mathcal{L}}$ von $\mathcal{L}$ ist definiert durch:

$$\varkappa_{\mathcal{L}}(\mathrm{w}) = \begin{cases} 0 & \text{falls w} \notin \mathcal{L} \\ 1 & \text{falls w} \in \mathcal{L} \end{cases}$$

Definition 1.4

Eine Folge $\mathrm{w} = \mathrm{w}_1, \ldots, \mathrm{w}_n$, $\mathrm{w}_i \in \sum$ heißt *Wort über* $\sum$, die Länge $l(\mathrm{w})$ ist n. Die Länge von ε ist 0.

Anders als in natürliche Sprache, gibt es in der THI immer auch eine Grammatik, welche die Regeln definiert, nach denen die Sprache gebildet wird:

Definition 1.5

Eine *Grammatik* G = (V, T, P, S) besteht aus den vier Teilen

V (Variable, auch Nichtterminale), das sind die Symbole, die innerhalb einer Ableitung, aber nicht im fertigen Wort, auftreten dürfen.

T (Terminale), das sind die Symbole, die im fertigen Wort auftauchen dürfen.

P (Produktionen), diese definieren die Regeln, nach denen die Ableitung vorgenommen werden darf.

S (Startsymbol), der Ausgangspunkt der Ableitung.

Beispiele für Grammatiken sind Ganzzahlen in Java, sowie korrekt geklammerte arithmetische Ausdrücke

Beispiel 1.2

$V = \{\text{S, D, Z}\}$

$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{-}\}$

$P = \{S \to DZ, D \to 0|1|2|3|4|5|6|7|8|9, Z \to DZ|\varepsilon\}$

$S = \text{S}$

Beispiel 1.3

$V = \{\text{S, E, T, F}\}$

$T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{-}, +, *, /, (,)\}$

$P = \{S \to E, E \to E + T|E - T|T, T \to T * F|T/F|F, F \to (E)|0|1|2|3|4|5|6|7|8|9\}$

$S = \text{S}$

Definition 1.6

Eine Grammatik heißt:

rechteslinear (regulär, Typ-3), wenn auf der linken Seite jeder Produktion genau ein Nicht-Terminal steht und auf der rechten Seite höchstens eines, zu dem muss es, falls es auftritt, das letzte Symbol sein. Zu den rechstlinearen Grammatiken gibt es mutatis mutandis auch die linklinearen Grammatiken.

linear , wenn auf der rechten Seite jeder Produktion höchstens ein Nichtterminal auftaucht (linke Seite wie gehabt).

kontextfrei, (Typ-2) wenn auf der rechten Seite eine beliebige Folge von Terminalen und Nichtterminalen steht (linke Seite wie gehabt).

kontextsensitiv, (Typ-1) wenn die rechte Seite nicht kürzer als die linke ist, wobei die letztgenannte ebenfalls eine beliebige nichtleere Folge von Terminalen und Nichtterminalen sein dar.

strukturiert, (Typ-0) wenn die rechte Seite beliebig und die linke Seite beliebig(aber nicht leer) sein kann.

Beispiel 1.4

Definition 1.7

Eine (nichtleere) Folge von Symbolen aus V ∪ T heißt *Wortform* wenn mindestens ein Symbol aus V enthalten ist. Besteht die Folge nur aus Elementen von T, so heißt sie ein *Wort*.

Definition 1.8

Eine Folge von Wortformen bzw. Worten $\mathrm{WF}_1, \ldots \mathrm{WF}_n$ heißt *Ableitung* des Wortes W, aus der Grammatik G, wenn:

- $\mathrm{WF}_1 = \mathrm{S}$ ist
- $\mathrm{WF}_n = \mathrm{W}$ ist
- $\mathrm{WF}_i = \mathrm{XLZ}$, $\mathrm{WF}_{i+1} = \mathrm{XRZ}$,
  $\mathrm{L} \to \mathrm{R} \in \mathrm{P}$

  Man schreibt in diesem Fall:

  $\mathrm{WF}_i \to \mathrm{WF}_{i+1}$, $S \overset{\mathrm{n}}{\to} W$, bzw. $S \overset{*}{\to} W$

Definition 1.9

Eine Ableitung heißt *Linksableitung* (nur bei Typ-2-Sprachen relevant), wenn immer das am weitesten links stehende Nichtterminal abgeleitet wird.

Definition 1.10

Die von einer Grammatik G *erzeugte Sprache* $\mathcal{L}_\mathrm{G}$ ist die Menge aller ableitbaren Worte.

Definition 1.11

Zwei Grammatiken G und G' heißen *äquivalent*, wenn $\mathcal{L}_\mathrm{G} = \mathcal{L}_\mathrm{G'}$.

Definition 1.12

Eine Produktion $\mathrm{L} \to \mathrm{R} \in \mathrm{P}$ heißt *Kettenproduktionen*, wenn R genau einem Nichtterminal besteht, sie heißt ε - Produktion, wenn $\mathrm{R} = \varepsilon$ ist.

Satz 1.1

Sei G eine Typ-2- oder Typ-3-Grammatik mit
$\varepsilon \notin \mathcal{L}_G$*. Dann gibt es eine äquivalente* ε*- freie Typ-2- oder Typ-3-Grammatik.*

Satz 1.1 Beweis

Man konstruiert die äquivalente Grammatik, indem man die ε- Produktion 'nach vorne zieht'. Anstelle einer Ableitung A $\rightarrow$ bC, gefolgt von C $\rightarrow \varepsilon$ leitet man direkt A $\rightarrow$ b ab.
Ggf. muss hier die Produktion A $\rightarrow$ b die Produktionsmenge hinzugefügt werden.
Das leitet der folgende Algorithmus:

- Ermittle Mengen der Variablen, aus denen ε ableitbar ist, d.h. x $\rightarrow \varepsilon \in P$, diese sei E.
- wiederhole, bis P stabil ist:
  Für alle Produktionen L $\rightarrow$ R in P:
  Wenn R eine Variable aus E enthält, dann füge die verkürzte Regel L $\rightarrow$ R' in P ein. Wobei R' aus R durch Streichung einer Variable aus E entsteht.
  Wenn R' $= \varepsilon$, dann füge L in E ein.
- Lösche alle ε - Produktionen aus P.

Bew.: Sei A $\rightarrow WF_1 \rightarrow WF_2 \rightarrow \ldots \rightarrow WF_n = w$ eine Ableitung unter der ursprünglichen Grammatik
Ann.: Der i-te Schritt sei $\mathrm{WF}_i = \mathrm{LXR} \rightarrow \mathrm{LR} = \mathrm{WF}_{i+1}$. Dann wurde X in einem früheren Ableitungsschritt aus einem V ein Y erzeugt, z.B. durch die Produktion Y - L' X R ! Aufgrund der Konstruktion gibt es nun eine Produktion Y $\rightarrow$ L' R'. Damit ist das Wort auch in der neuen Grammatik

Rückrichtung:
Ann.: Sei A $\rightarrow WF_1 \rightarrow WF_2 \rightarrow \ldots \rightarrow WF_n = w$ eine Ableitung unter der neuen Grammatik. Wir verwenden im Schritt $WF_i \rightarrow WF_{i+1}$ eine Regel(Produktion), die so in der alten Grammatik nicht existierte. Dann existiert eine Regel mit zusätzliche Variable auf der rechten Seite, die alle direkt oder indirekt zu ε ableitbar waren. Damit ist die Ableitung auch in der alten Grammatik möglich.

Eine andere Technik, die unter Umständen hilfreich ist, betrift die Elimination von Kettenproduktionen:

Satz 1.2

Sei G = (V, T, P, S) eine Typ-2- oder Typ-3-Grammatik mit Kettenproduktionen. Dann gibt es eine zu G äquivalente Typ-2- oder Typ-3-Grammatik ohne Kettenproduktionen.

Satz 1.2 Beweis

- Ermittle alle Kettenproduktionen, d.h. Produktionen, die die Form A $\rightarrow$ B C haben, wobei B und C Variablen sind. Diese sei K.
- Füge für jede Kettenproduktion A $\rightarrow$ B C die Produktionen A $\rightarrow$ B' und B' $\rightarrow$ C in P ein.
- Lösche alle Kettenproduktionen aus P.

2 Grammatiknormalformen ÜBERSPRUNGEN

3 Reguläre Sprachen und endliche Automaten

Wir wechseln von den allgemeinen Grammatiken zu den einfachsten Sprachen, den damit verbundenen Grammatiken und den für die Erkennung notwendingen Mechanismen.Die typische Anwendung besteht nicht in der Erzeugung von Werten, sondern um ihre Erkennung.
Als Beispiel betrachten wir den javac:

- Prüfe, ob der Wert zur Sprache gehört
- Falls ja, übersetze das Wort in die Zielsprache unter Nutzung der Information aus der Ableitung
- Falls nein 'make an educated guess', welcher Fehler aufgetreten ist, und setze die Übersetzung so gut wie möglich fort.

Die einfachsten Sprachen der Chomsky-Hierarchie sind die regulären oder Typ-3-Sprachen, diese haven vielfältige Anwendungen.

- Reguläre Ausdrücke zur Beschreibung von Programmiersprachen (z.B. in den JLS, Pro1, ...)
- Entprellen von Tastaturen mit Rollover und Wiederholung
- Erkennung von Barcodes und Dotcodes mit Hw. unbekannte Tastrate und Auflösung

Betrachten wir das Problem der Tastaturentprellung mit Wiederholung und Rollover.
Zur Lösung dieses Problems mit minimalen Speicherplatz reicht 1 Byte pro Taste aus.

- Wird eine Taste gedrückt, schließt und öffnet der Kontakt mehrmals kurz hintereinander. Die Steuerung muss eine bestimmte Zeit abwarten, bevor das 1. Zeichen erkannt wird.
- Nach einer weiteren Wartezeit beginnt die Wiederholung
- Wird dann eine andere Taste gedrückt, so gerät die erste Taste in einen Sperrzustand, d.h. die Wiederholung startet nicht oder stopp.

Nebenbedingungen:

- kein Busy Wait, d.h. zwischen zwei Tastenabfragen steht der Mikro-Controller für andere Aufgaben zur Verfügung.
- Nur sehr wenig RAM, z.B. 512 Byte.

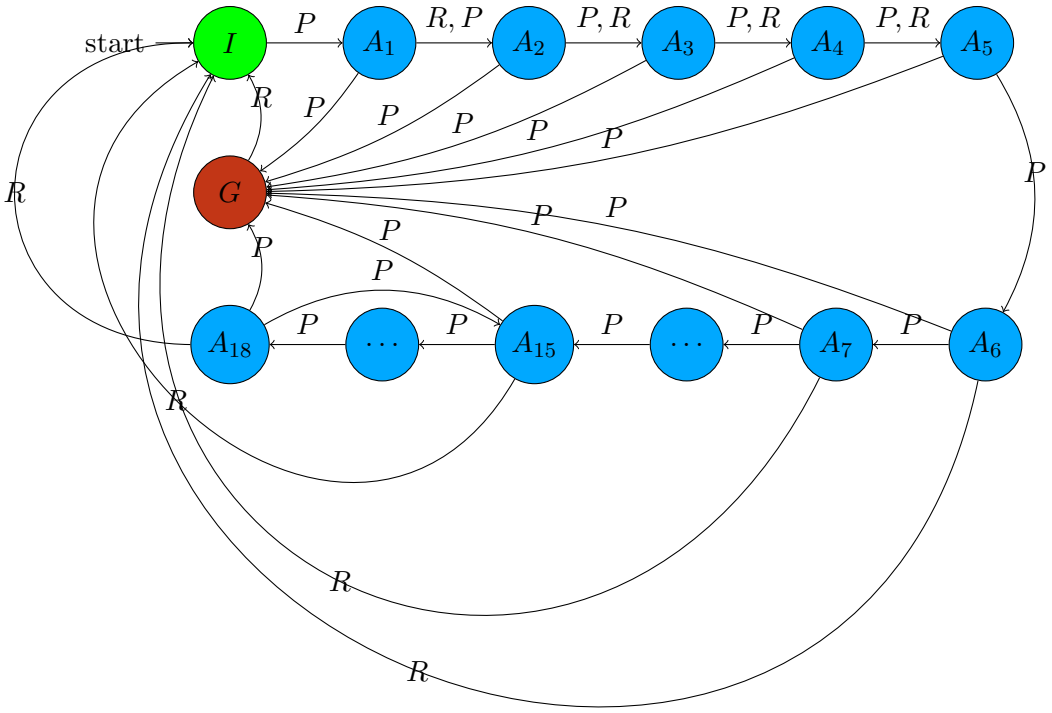
Relevant sind 3 Ereignisse:

- Taste gedrückt $\Longrightarrow$ P
- Taste loslassen $\Longrightarrow$ R
- Andere Taste gedrückt $\Longrightarrow$ O

Wir erstellen ein Modell mit mehreren Zuständen:

- Inaktiv: taste loslassen
- Gesperrt: Taste war aktiv, andere Taste später gedrückt

- Aktiv_x: Taste wurde vor x Zyklen gedrückt



3.1 Endliche Automaten

Allgemein lässt sich ein endlicher Automat wie folgt definieren:

- Als Diagramm mit beschrifteten Übergängen

  einem Startzustand, markiert durch start ⟶○

  einem oder mehrere Engzustünden, markiert durch ◎

- Durch Angabe der Zustände, Eingaben, Übergangsfunktionen und Zielzustände, dies führt zur folgenden Definition:

Definition 3.1

Ein *endlicher Automat* ist ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$, wobei

- Q eine endliche Menge von Zuständen ist
- Σ eine endliche Menge von Eingaben ist
- $\delta : Q \times \Sigma \to Q$ ist eine Zustandsübergangsfunktion
- $q_0 \in Q$ ist ein Startzustand
- $F \subseteq Q$ ist eine Menge von Endzuständen

Gemäß Def. 3.1 kann ein Automat nur einzelne Zeichen verarbeiten, wir verallgemeinern daher δ wie folgt:

Definition 3.2

Für einen endlichen Automaten ist die erweiterte Zustandsübergangsfunktion $\hat{\delta}$ definiert durch:

$$\hat{\delta}_n(q, w_1 \dots w_n) = \begin{cases} q, & \text{falls } n = 0 \ (w = \varepsilon) \\ \delta(q, w_1), & \text{falls } n = 1 \\ \delta(\hat{\delta}_{n-1}(q, w_1, \dots w_{n-1}, w_n)) & \text{falls } n \geq 2 \end{cases}$$

Für ein beliebiges w $\in \Sigma^*$ mit $|\mathrm{w}| = m$ definieren wir $\hat{\delta}(q, w) = \hat{\delta}_m(q, w)$.

Oft ist δ und damit auch $\hat{\delta}$ nicht total, d.h. für eine Kombination aus (q, w) ist kein Nachfolgezustand definiert. In diesen Fall 'hängt' der Automat, d.h. er befindet sich in 'keinem' Zustand.

Definition 3.3

Für einen endlichen Automaten $\mathrm{M} = (\mathrm{Q}, \Sigma, \delta, \mathrm{q_0}, \mathrm{F})$ ist die von M *akzeptierte Sprache* $\mathcal{L}_\mathrm{M}$ definiert durch:

$$\mathcal{L}_\mathrm{M} = \{w \in \Sigma^* \mid \hat{\delta}(\mathrm{q_0}, w) \in \mathrm{F}\}$$

Mit anderen Worten: Gerät der Automat in einen Zustand aus Q\F oder hängt er, wird das Wort nicht akzeptiert.
Der Automat akzeptiert also alle Wörter, die in einem Endzustand enden.

Satz 3.1

Zu jeder Typ-3-Grammatik G = (V, T, P, S) gibt es einen endlichen Automaten $M = (Q, \Sigma, \delta, q_0, F)$*, so dass* $\mathcal{L}_G = \mathcal{L}_M$ *und umgekehrt.*

Wir benötigen für den Beweis noch ergänzende Definitionen um ihn in beiden Richtungen zeigen zu können.
Zunächst zeigen wir, dass $\mathcal{L}_G = \mathcal{L}_M$ ist
Dies funktioniert wie folgt:

- Setze V = Q, d.h. für jeden Zustand, q e x eine Variable v.
- Setze T = Σ

- Setze P = $\{V_{q1} \to t\ V_{qz} \mid \delta(q_1, t)\} \cup \{V_{q1} \to t \mid \delta(q_1, t)\}$
- S = V_{q0}

Falls w = $w_1 \ldots w_n \in \mathcal{L}_M$, so ist $\hat{\delta}(q_0, w) = q_x$ gilt, gilt dann wenn es ein aus V_{q0} ableitbare Wortform $w_1, w_2 \ldots w_n\ V_{qx}$ gibt. Wegen der Definition von P ist aber, falls $q_x \in F$ liegt, auch die Produktion vorhanden, die w_n ohne V_{qx} erzeugt.
Der umgekehrte Weg ist komplex, da:

- Grammatiken in einen Schritt beliebig viele Terminale erzeugen können:
  A → bedef G. Das ist durch eine lokale Änderung der Produktionen erreichbar:
  $A \to bA', A \to cA'', A'' \to dA''', A''' \to eA'''', A'''' \to fG$, wobei A', A" ... in keine andere Produktionen vorkommen dürfte.
- Grammatiken können Kettenproduktionen enthalten (A → B), diese können nach Satz 1.2 eliminiert werden.
- Ableitungen enden typisch mit einer Produktion A → b. Dort ersetzen wir die Produktion durch etwas offensichtlich äquivalenten: $A \to bA', A' \to \varepsilon$.
- ε-Produktionen werden zu Endzuständen, d.h. wenn x → ε in P ist, so ist $V_x \in F$.
- Grammatiken sind ggf. nicht eindeutig, z.B. bei der Erkennung ganzer Zahlen in Java:
  Z → 0 < irgendwas oktales > | 0x < irgendwas oktales > | 0b < irgendwas binäres > | 1-9 < irgendwas dezimales >

Wir unterbrechen den Beweis von Satz 3.1 daheim definieren

Definition 3.4

Ein *nichtdeterministischer endlicher Automat* M = (Q, Σ, δ, q_0, F) ist eine Verallgemeinerung (deterministischen) endlichen Automaten, mit folgenden Unterschieden:

- $\delta : Q \times \Sigma \to P(Q)$ definiert für eine Kombination aus Zustand und Zeichen mehrere mögliche Nachfolgezustände
- δ ist total, indem ggf. $\delta(q_i, i) = \emptyset$ gilt.

Ebenso müssen auch noch die Definition der erweiterten Zustandsübergangsfunktion und der der Akzeptanz angepasst werden.