Algorithmen und Datenstrukturen

Theoretische Grundlagen der Informatik

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Algorithmen und Datenstrukturen

Sprachen zur Codierung von algorithmischen Problemen

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Standortbestimmung

Standort

- **Wir haben gelernt**
  - wie man auch komplexere Daten durch Wörter (Sprachen) beschreiben kann
  - dass Σ_{Bool} als Alphabet ausreichend ist (Homomorphismen)
  - Wie viele Wörter enthält eine Sprache, wenn Sie nicht unendlich ist?
- **Noch zu klären**
  - Wie kodieren wir algorithmische Probleme mithilfe von Sprachen?
    [Das sind Probleme, die wir durch Algorithmen lösen wollen]
- **Offen gebliebene Fragen zur Kodierung von Daten**
  - Gibt es eine zu einem Datum eine kürzeste Darstellung als Wort über einem beliebigem Alphabet?
    [Diese Frage hatten wir vertagt]

Algorithmus

- **Ziel**: Klassen von Problemen kennen und verstehen
  Probleme, die durch Algorithmen lösbar sind
- **Es fehlt**: formale Definition eines Algorithmus

Definition 1.31 (Algorithmus - Provisorium 2)

Seien Σ_1 und Σ_2 Alphabete und $\mathcal{D} \subseteq \Sigma_1^*$. Ein **Algorithmus** A ist eine totale Funktion

$$A : \mathcal{D} \to \Sigma_2^*$$

Zwei Algorithmen $A : \mathcal{D} \to \Sigma_2^*$ und $B : \mathcal{D} \to \Sigma_2^*$ sind genau dann **äquivalent** (in Zeichen $A \equiv B$), wenn

$$\forall x \in \mathcal{D} \ : \ A(x) \ = \ B(x)$$

gilt.

- **Beachte**:
  - Ein- und Ausgaben sind als Wörter codiert
  - A bestimmt für jede gültige Eingabe x eine eindeutige Ausgabe $A(x)$
    A ist **total korrekt**

Übersicht

Wir betrachten folgende Problemklassen

- Entscheidungsprobleme / Wortprobleme
- Äquivalenzprobleme
- Funktionsprobleme
- Relationsprobleme
- Optimierungsprobleme

Wir werden das folgende wichtige Ergebnis erhalten:

! Wort-, Äquivalenz-, Funktions-, Relations- und Optimierungsprobleme lassen sich auf **Entscheidungsprobleme** zurückführen

Entscheidungsproblem / Wortproblem

Definition 1.32 (Entscheidungsproblem / Wortproblem)

Das **Entscheidungsproblem** $\langle \Sigma, L \rangle$ für ein gegebenes Alphabet Σ und eine Sprache $L \subseteq \Sigma^*$ ist, für jedes $w \in \Sigma^*$ zu entscheiden, ob

$$w \in L \quad \text{oder} \quad w \notin L$$

daher spricht man auch vom **Wortproblem** $\langle \Sigma, L \rangle$.
Ein Algorithmus $A : \Sigma^* \to \{0, 1\}$ löst das Entscheidungsproblem $\langle \Sigma, L \rangle$, falls für alle $w \in \Sigma^*$ gilt:

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

Wir sagen auch: A **erkennt** L.

Beachte: Die wichtigste Problemklasse für die **Berechenbarkeitstheorie**! Wir werden sehen, dass sich andere Problemklassen auf Entscheidungsprobleme zurückführen lassen.

Beispiele für Entscheidungsprobleme 1

- Entscheidungsprobleme können sehr komplex sein
- Betrachten wir daher ein paar Beispiele

Beispiel 1.27 (Primzahltest)

Ein wichtiges Entscheidungsproblem ist der **Primzahltest**

$$\langle \Sigma_{Bool}, \{w \mid Zahlwert(w) \text{ ist eine Primzahl}\}\rangle$$

Beispiel 1.28 (Erfüllbarkeitsproblem)

Das **Erfüllbarkeitsproblem** $\langle \Sigma_{logic}, \mathrm{ERF}\rangle$ ist ein Entscheidungsproblem, wobei

$$\mathrm{ERF} = \left\{ w \in \Sigma_{logic}^{*} \;\middle|\; w \text{ kodiert erfüllbare Formel} \right\}$$

Beispiele für Entscheidungsprobleme 2

Beispiel 1.29 (Problem des Hamiltonschen Kreises)

Das **Problem des Hamiltonschen Kreises** ist $\langle \Sigma, HK \rangle$ mit $\Sigma = \{0, 1, \#\}$ und

$HK = \{w \in \Sigma^* \mid w \text{ kodiert ungerichteten Graphen, der Hamiltonschen Kreis enthält}\}$

Beispiel 1.30 (Äquivalenzproblem)

Das **Äquivalenzproblem** $\langle \Sigma_{Tastatur}, \{B \in \Sigma^*_{Tastatur} \mid B \text{ ist äquivalent zu } A\} \rangle$ bedeutet zu entscheiden, ob zwei Algorithmen A und B auf identischem Eingabealphabet Σ arbeiten und ob gilt:

$$\forall w \in \Sigma^* : A(w) = B(w)$$

Rekursive Sprachen

Definition 1.33 (rekursive Sprache)

Sei Σ ein Alphabet und $L \subseteq \Sigma^*$ eine Sprache. L heißt genau dann **rekursiv**, wenn es einen Algorithmus A gibt, der das Entscheidungsproblem $\langle \Sigma, L \rangle$ löst.

Anmerkungen:

- **wichtig**: A terminiert für jede Eingabe!
- rekursive Sprachen nennt man auch **entscheidbare Sprachen**

Aufzählprobleme

Definition 1.34 (Aufzählproblem / Aufzählungsalgorithmus)

Seien Σ ein Alphabet, $L \subseteq \Sigma^*$ eine Sprache und $n \in \mathbb{N}$. Das **Aufzählproblem** $\langle \Sigma, L, n \rangle$ für L besteht darin, die kanonisch ersten n Wörter von L aufzulisten.

$A : \mathbb{N} \to \mathcal{P}(L)$ ist ein **Aufzählungsalgorithmus für** L, falls A für jede Eingabe $n \in \mathbb{N}$ die kanonisch ersten n Wörter $w_0, w_1, \ldots, w_{n-1}$ der Sprache L ausgibt.

Beachte: Mit der kanonischen Ordnung können wir für jedes Σ^* leicht einen Aufzählungsalgorithmus finden.

Aufzählbarkeit und Entscheidbarkeit 1

Aufzählbarkeits- und Entscheidungsprobleme hängen eng zusammen:

Satz 1.1

Seien Σ ein Alphabet und $L \subseteq \Sigma^*$. L ist genau dann rekursiv, wenn es einen Aufzählungsalgorithmus für L gibt.

Beweis.

Ist L endlich, so ist die Aussage trivial. Sei L also unendlich.

- L rekursiv $\Rightarrow$ Aufzählungsalgorithmus für L existiert.
  Sei A der Entscheidungsalgorithmus für L, dann erhalten wir einen Aufzählungsalgorithmus $B(n)$ für L wie folgt:

```
B(n) ::=
  result = () /* leere Folge */
  foreach w ∈ Σ* in canonical order
    if size(result) = n then return result
    if A(w) then result = result + w
  end foreach
```

Aufzählbarkeit und Entscheidbarkeit 2

Beweis (Fortsetzung).

- Aufzählungsalgorithmus für L existiert $\Rightarrow$ L rekursiv.
  Sei $B(n)$ der Aufzählungsalgorithmus für L, dann gewinnt man einen Entscheidungsalgorithmus $A(w)$ für L wie folgt:

```
A(w) ::=
  n = 0;
  while true
    /* Kanonisch n ersten Wörter berechnen */
    (v_1, ..., v_n) = B(n)
    if (v_n = w)  return 1
    if (|v_n| > |w|) return 0
    n = n + 1
  end while
```

□

Funktionsprobleme

Definition 1.35 (Funktionsproblem / berechenbar)

Seien Σ_D und Σ_W zwei Alphabete. Wir sagen, dass ein Algorithmus A eine partielle Funktion (Transformation) $f : \Sigma_D^* \dashrightarrow \Sigma_W^*$ berechnet (oder das **Funktionsproblem** f löst), falls:

$$\forall w \in \mathrm{dom}(f) : A(w) = f(w)$$

Existiert ein Algorithmus, der f berechnet, so nennen wir f **berechenbar**.

Anmerkungen:

- Damit: **Intuitiver Berechenbarkeitsbegriff**
- f ist partielle Funktion (hat u.U. Definitionslücken, d.h. $\mathrm{dom}(f) \neq \Sigma_D^*$)
- Es bleibt offen, wie Algorithmus bei Eingabe von Definitionslücken reagiert

! Funktionsprobleme kann man in Entscheidungsprobleme überführen!

Lemma 1.1

Seien $f : \Sigma_D^* \dashrightarrow \Sigma_W^*$ ein Funktionsproblem und $encode : \Sigma_D^* \times \Sigma_W^* \to \Sigma_{Bool}^*$ eine injektive Funktion. Ferner sei

$$L_f = \{encode\,(x, f\,(x)) \mid x \in dom(f)\}$$

eine Sprache. Dann gilt

$$\langle \Sigma_{Bool}, L_f \rangle \text{ entscheidbar} \quad \Leftrightarrow \quad f \text{ berechenbar}$$

Funktionsprobleme und Entscheidungsprobleme 2

Beweis.

- $\langle \Sigma_{Bool}, L_f \rangle$ entscheidbar $\rightarrow$ f berechenbar: Sei A der Entscheidungsalgorithmus für L_f - dann gewinnen wir wie folgt einen Algorithmus B zur Berechnung von f :

```
B(x) ::=
      foreach w ∈ Σ*_W in canonical order
        if A( encode(x,w) ) return w
      end foreach
```

Anmerkung: Das Verfahren ist zwar nicht effizient, aber effektiv!

effektiv vs. effizient

Zur Erinnerung:

- **effektiv** bedeutet wirkungsvoll im Verhältnis zu den aufgewendeten Mitteln
- **effizient** bedeutet leistungsfähig, wirtschaftlich

Funktionsprobleme und Entscheidungsprobleme 3

Beweis (Fortsetzung).

- f berechenbar $\rightarrow$ $\langle \Sigma_{Bool}, L_f \rangle$ entscheidbar: Sei B der Algorithmus, der f berechnet. Einen Entscheidungsalgorithmus A für L_f gewinnt man wie folgt:

  ```
  A(w) ::=
    (x, y) = encode^-1(w)
    if B(x) = y return 1 else return 0
  ```

  $\square$

Optimierungsprobleme

Mit Sprachen lassen sich auch komplexere Problemstellungen beschreiben:

Definition 1.36 (Optimierungsproblem)

Ein **Optimierungsproblem** ist ein 6-Tupel $\mathcal{U} = (\Sigma_{In}, \Sigma_{Out}, L, \mathcal{M}, cost, goal)$, wobei

- Σ_{In} ist ein Alphabet (das **Eingabealphabet**)
- Σ_{Out} ist ein Alphabet (das **Ausgabealphabet**)
- $L \subseteq \Sigma_{In}^*$ ist die Sprache der **zulässigen Eingaben**. $x \in L$ heißt **Instanz** von $\mathcal{U}$
- $\mathcal{M} : L \to \mathcal{P}(\Sigma_{Out}^*)$ ist eine Funktion - für jedes $x \in L$ ist $\mathcal{M}(x)$ die Menge der **zulässigen Lösungen** für x.
- $cost : \bigcup_{x \in L} (\mathcal{M}(x) \times \{x\}) \to \mathbb{R}^+$: ist eine Funktion, genannt **Kostenfunktion**
- $goal \in \{Minimum, Maximum\}$ ist das **Optimierungsziel**

Eine zulässige Lösung $a \in \mathcal{M}(x)$ heißt **optimal** für die Instanz x des Optimierungsproblems $\mathcal{U}$, falls

$$\mathrm{Opt}_{\mathcal{U}}(x) = cost(a, x) = goal\,\{cost(\beta, x) \mid \beta \in \mathcal{M}(x)\}$$

Optimierungsprobleme

Definition 1.36 (Optimierungsproblem - Fortsetzung)

Ein Algorithmus A löst $\mathcal{U}$, falls für jedes $x \in L$

1. $A(x) \in \mathcal{M}(x)$; d.h. $A(x)$ ist eine zulässige Lösung der Instanz x von $\mathcal{U}$.
2. $cost(A(x), x) = goal\, \{cost(\beta, x) \mid \beta \in \mathcal{M}(x)\}$.

Falls $goal = Minimum$, sprechen wir von einem **Minimierungsproblem**; falls $goal = Maximum$ sprechen wir von einem **Maximierungsproblem**.

Beispiel 1.31 (Das Traveling Salesman Problem)

Das Traveling Salesman Problem ist ein Optimierungsproblem:

- $\Sigma_{In} = \Sigma_{Out} = \{0, 1, \#\}$ - gewichtete Graphen und Pfade kodieren wir, wie oben beschrieben
- $L = \{w \in \Sigma_{In}^* \,|\, w \text{ kodiert vollständigen, schleifenfreien Graphen}\}$
- $\mathcal{M}(w) = \{v \in \Sigma_{Out}^* \mid v \text{ kodiert Hamiltonschen Kreis in } w\}$
- $cost(v, w) = \phi(p)$, falls v den Pfad p und G den Graph $\langle V, E, \phi \rangle$ kodiert
- $goal = Minimum$

Optimierungsprobleme als Entscheidungsprobleme

! Optimierungsprobleme kann man in Entscheidungsprobleme überführen!

Wir betrachten hier exemplarisch das Traveling Salesman-Problem:

- $\mathcal{M}(w)$ zu berechnen, bedeutet das Entscheidungsproblem $\langle \Sigma_{Out}, L_{Hamil} \rangle$ zu lösen, wobei

$$L_{Hamil} = \{ v \mid v \text{ ist Hamiltonscher Kreis in } G \}$$

  ist

- $Opt_{\mathcal{U}}(w)$ zu berechnen, bedeutet das Entscheidungsproblem $\langle \Sigma_{Out}, L_{HamilOpt} \rangle$ zu lösen, wobei

$$L_{HamilOpt} = \left\{ v \;\middle|\; \phi(v) = Minimum \bigcup_{v' \in L_{Hamil}} \phi(v') \right\}$$

Standortbestimmung

Standort

- **Wir haben gelernt**
  - wie man auch komplexere Daten durch Wörter (Sprachen) beschreiben kann
  - dass Σ_{Bool} als Alphabet ausreichend ist (Homomorphismen)
  - wie man die Mächtigkeit (Größe) einer Sprache bestimmt
  - wie man mit Sprachen algorithmische Probleme beschreibt
  - dass sich auch komplexe Probleme auf **Entscheidungsproblem** zurückführen lassen

Neue Fragestellung

Wie kann man Algorithmen zur Lösung von **Entscheidungsproblemen** formal (systematisch) beschreiben?