First-Longest-Match-Analyse

Niklas Rieken

28. Januar 2018

Wir schauen uns hier einmal eine praktische Anwendung von regulären Ausdrücken und endlichen Automaten an. Bisher haben wir stets das *einfache Matching-Problem* für reguläre Ausdrücke betrachtet. Dabei ging es darum zu prüfen, ob ein gegebenes Wort $w \in \Sigma^*$ zur Sprache eines regulären Ausdrucks $r \in \mathsf{RE}_\Sigma$ gehört. Dies ließ sich mit Hilfe der Thompson-Konstruktion einfach als das Wortproblem eines ε-NFA betrachten. In der Praxis ist das einfach Matching-Problem aber zu primitiv um echte Probleme lösen zu können. Vorallem im Compilerbau benötigen wir etwas mehr, wenn wir Programmcode in seine Bestandteile zerlegen wollen um später Syntax-Checks darauf durchführen zu können und schließlich eine Semantik für das Programm festzulegen. In der Fachsprache nennt man diesen Teil eines Compilers auch *Scanner* oder *Lexer* (siehe auch die Programme `lex`, `flex`). In diesem Dokument verwenden wir die Variablen h, i, j, k, ℓ stets als natürliche Zahlen aus einer Menge $[m] \coloneqq \{1, \dots, m\}$. Manchmal ist die Notation etwas sloppy, aber hoffentlich verständlich genug.

Problemstellung: Extended Matching Problem

Das *erweiterte (extended) Matching-Problem* kommt direkt aus der Anwendung des Compilerbaus. Gegeben ist ein Wort $w \in \Sigma^*$ und eine Reihe von regulären Ausdrücken über Σ $r_1, \dots, r_n \in \mathsf{RE}_\Sigma$. (*Bemerkung:* In der Praxis ist die Annahme $\varepsilon \notin L(r_i) \neq \varnothing$ für alle i sinnvoll). Gesucht ist nun eine Zerlegung des Wortes $w = u_1 \dots u_k$, wobei für jedes u_j ein r_{i_j} existieren muss, sodass $u_j \in L(r_{i_j})$. Man nennt die Zerlegung in $(u_1, \dots, u_k)$ auch *Dekomposition* und die zugehörigen Indizes der regulären Ausdrücke $(i_1, \dots, i_k)$ *Analyse* von w bezüglich $r_1, \dots, r_n$.
Wir sehen schnell, dass weder Dekomposition, noch Analyse eindeutig sein müssen (insbesondere dann, wenn wir $\varepsilon \in L(r_i)$ zulassen).

Beispiel 1. (i) $r_1 = a^+, w = aa$. Ergibt Dekompositionen (aa) und (a, a) mit zugehöriger (eindeutiger) Analyse (1) bzw. $(1, 1)$.

(ii) $r_1 = a + b, r_2 = a + c, w = a$. Ergibt eindeutige Dekomposition (a) mit zwei verschiedenen Analysen (1) und (2).

Im zweiten Fall stellen wir uns vor, dass r_1 mögliche Schlüsselwörter einer Programmiersprache (`while`, `true`, `if`) beschreibt und r_2 mögliche Variablennanem (Identifier), dann sieht man warum eine eindeutige Analyse wichtig ist. In diesem Anwendungsfall ist es natürlich zu sagen, dass der Ausdruck r_1 "wichtiger" ist als r_2 (deshalb verbieten die mesiten Programmiersprachen auch Schlüsselwörter als Identifier). Ähnliche Beispiele lassen sich auch für die Dekomposition finden.

Wir wollen nun sowohl, die Dekomposition, als auch die Analyse eindeutig machen. Ein erster Ansatz wäre zunächst das leere Wort zu verbieten und sicherzustellen, dass $L(r_i) \cap L(r_j) = \varnothing$ für alle $i \neq j$, indem wir den gemeinsamen Schnitt von zwei Ausdrücken aus dem "unwichtigeren" Ausdruck entfernen. Dadurch würde zumindest die Analyse eindeutig werden. Diese Idee ist jedoch nicht sinnvoll, u.a. deshalb, da das Entfernen des Schnittes – also das Erkennen von $L(r'_j) := L(r_j) \setminus L(r_i)$ – eine Produktkonstruktion erfordert und somit tendenziell teuer ist.

Eindeutigkeit durch First-Longest-Match

Der am meisten verbreitete Weg ist es folgende zwei Prinzipien zu implementieren:

Longest Match Mache jedes u_i der Zerlegung so lang wie möglich.

First Match Wähle aus den matchenden regulären Ausdrücken den mit dem kleinesten Index.

Definition 2. Eine Dekomposition $(u_1, \ldots, u_k)$ von $w \in \Sigma^*$ bezüglich der regulären Ausdrücke $r_1, \ldots, r_n \in \mathsf{RE}_\Sigma$ heißt *Longest-Match-Dekomposition* (LMD), wenn für jedes $i \in [k]$, $x \in \Sigma^+$, $y \in \Sigma^*$ mit $w = u_1 \ldots u_i x y$ gilt, dass kein $j \in [n]$ existiert, sodass $u_i x \in L(r_j)$.

Umgangssprachlich bedeutet das, dass wir an einen Teil der Komposition u_i kein nichtleeres Wort mehr anhängen können, was noch nicht verarbeitet wurde. Es ist klar, dass eine LMD eindeutig ist, falls sie existiert. Sie muss jedoch nicht immer existieren:

Beispiel 3. $r_1 = a^+, r_2 = ab, w = aab$ hat Dekomposition (a, ab) aber keine LMD.

Definition 4. Sei $(u_1, \ldots, u_k)$ eine LMD von $w \in \Sigma^*$ bezüglich der regulären Ausdrücke $r_1, \ldots, r_n \in \mathsf{RE}_\Sigma$. Die zugehörige *First-Longest-Match-Analyse* (FLM-Analyse) $(i_1, \ldots, i_k)$ ist gegeben durch

$$i_j := \min\{\ell \mid u_j \in L(r_\ell)\}$$

für jedes $j \in [k]$.

Auch hier ist klar, dass es höchstens eine FLM-Analyse gibt und sie existiert, wenn die LMD existiert.

Eine mögliche Implementierung

Es gibt viele Möglichkeiten eine FLM-Analyse durchzuführen. Die von mir hier vorgestellte Art ist nicht die von mir favorisierte, aber sie ist insofern intuitiv, als dass sie nur Konzepte aus der FoSAP-Vorlesung verwendet und ein wenig über Arrays.

Algorithmus 1 : First-Longest-Match-Analyse

<u>FLM</u>$(w, (r_i)_{i=1}^n)$

Output : FLM-Analyse $(i_j)_{j=1}^k$ und LMD $(u_j)_{j=1}^k$

`// Vorbereitung`

Wandle alle r_i mit Thompson-Konstruktion in ε-NFAs $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ um.

Baue neuen ε-NFA $\mathcal{A} = (\biguplus_i Q_i \uplus \{q_0, q_f\}, \Sigma \uplus [n], \delta, q_0, \{q_f\})$ mit $\delta(p, a) = \delta_i(p, a)$ falls $p \in Q_i$ und $\delta(q_0, \varepsilon) = \{q_0^i \mid i \in [n]\}$ und $\delta(q, i) = \{q_f\}$, falls $q \in F_i$.

Eliminiere ε-Transitionen.

Determinisiere mit Potenzmengenkonstruktion.

Minimiere mit Markierungsalgorithmus.

`/* Wir haben jetzt einen DFA mit folgender Eigenschaft: Vom Zustand` $\hat{\delta}(q_0, u)$ `ist genau dann eine Transition mit` $i \in [n]$ `in einen Endzustand möglich, wenn` $u \in L(r_i)$`. */`

`// Löse nun das erweiterte Matching-Problem`

Setze $\ell = 1, w_\ell = w$.

while $w_\ell \neq \varepsilon$ **do**

- `A` = leeres Array der Länge $|w_\ell|$.
- Simuliere $\mathcal{A}$ auf w_ℓ, prüfe in jedem Schritt j, ob eine i-Transition in einen akzeptierenden Zustand möglich ist. Falls ja, setze `A[`j`]` auf das kleinste mögliche i.
- Nach der Simulation laufe `A` von hinten nach vorne durch bis wir den ersten nicht-leeren Eintrag an Stelle h finden. Sollte es keinen geben gebe einen Error aus, ansonsten ist $u_\ell = w_{\ell_1} \dots w_{\ell_h}$ und $i_\ell =$ `A[`h`]`.
- Setze $w_{\ell+1} = w_{\ell_h} \dots w_{\ell_{|w_\ell|}}, \ell = \ell + 1$.

`// Jetzt ist` $w_\ell = \varepsilon$`.`

Gebe $(u_1, \dots, u_{\ell-1})$ und $(i_1, \dots, i_{\ell-1})$ aus.

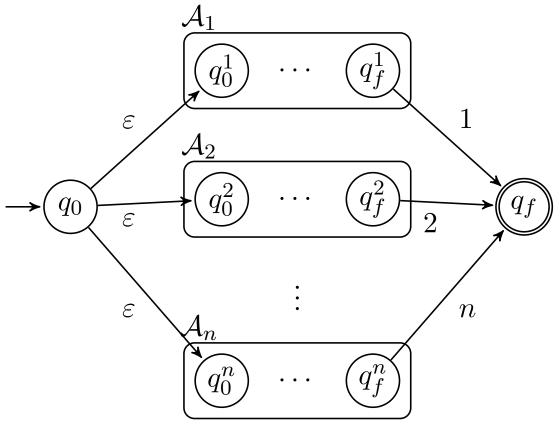

Abbildung 1: Skizze des NFAs aus Algorithmus 1.

Als Eingabe erhalten wir also $w \in \Sigma^*$ und die regulären Ausdrücke $r_1, \ldots, r_n \in \mathsf{RE}_\Sigma$. Gesucht ist eine FLM-Analyse inklusive zugehöriger LMD oder ein Error, falls diese nicht existieren.
In Abbildung 1 seht ihr wie der ε-NFA der nach Zeile 3 entsteht aussieht.