Projekte zur Vorlesung

Theoretische Informatik I

Prof. Dr. Christoph Kreitz / Mario Frank
Universität Potsdam, Theoretische Informatik, WS 2017/2018

Bearbeitungszeit: 4 Wochen nach Annahme des Projektes

Um die Verknüpfung zwischen der Theorie und Praxis zu verbessern und Ihnen die Möglichkeit zu geben, Anwendungsgebiete der Themen der Theoretischen Informatik I kennen zu lernen, bieten wir dieses Semester kleine Implementierungs-Projekte an. Sie können eines dieser Projekte bearbeiten und sich dadurch 10 % der Klausurpunkte als Bonus verdienen.
Die Projekte sind im allgemeinen einzeln zu bearbeiten, das heißt, Gruppenabgaben sind nur erlaubt, wenn dies explizit definiert ist. Bei allen Projekten handelt es sich um Implementierungsprojekte für Verfahren, die in der Vorlesung vorgestellt wurden.
Für alle Projekte muss neben den Quellcodes auch eine Dokumentation eingereicht werden, die die Funktionsweise des Programms beschreibt und ausreichend begründet, wieso die Implementierung korrekt ist. Es ist nicht notwendig, einen Beweis zu führen, wenn die Begründung detailliert genug ist, um die Äquivalenz zum Verfahren aus der Vorlesung zu begründen.

1 Allgemeine Definitionen

Bei allen Aufgaben, die einen Automaten behandeln sollen, können die Zustände S (Start), F (Final), N (Normal) oder SF (Start und Final) sein.
Als OCaml-Definition für den Zustandstypen wird dann die folgende verwendet.

```
type state_type = S | F | N | SF;;
```

2 Aufgaben

Aufgabe 1 (Teilmengenkonstruktion - 1 Person)

Implementieren Sie die (nicht optimierte) Teilmengenkonstruktion für NEAS mit zwei Symbolen (a und b) in OCaml. Als Eingabe soll Ihr Programm eine Zustands-Übergangstabelle erwarten. Diese wird als Liste von 4-Tupeln übergeben, wobei das erste Element der Zustandstyp, das zweite der Quell-Zustand und die letzten zwei Elemente des 4-Tupels die Listen der Nummern der Ziel-Zustände sind. Der Zustandstyp ist in den allgemeinen Definitionen festgeschrieben. Ist für einen Zustand des NEA ein Übergang mit einem Symbol nicht definiert, so ist der Zielzustand die leere Liste.
Die Definition der Eingabe ist in OCaml also

```
type nfa_transition = state_type * int * int list * int list;;

type nfa_transition_table = nfa_transition list;;
```

Als Ausgabe wird die Zustandsübergangstabelle des generierten DEA erwartet. Diese soll eine Liste von 5-Tupeln sein. Das erste Element ist ein Boolescher Wert, der aussagt, ob der Zustand erreichbar ist. Der zweite Wert ist der Typ des Zustands. Die restlichen drei Elemente sind Listen von Zuständen.
Die Definition in OCaml ist also

```
type dfa_transition = bool * state_type * int list * int list * int list;;

type dfa_transition_table = dfa_transition list;;
```

Stellen Sie sicher, dass die Listen von Zuständen jeden Zustand nur einmal enthalten und die Elemente der Listen aufsteigend sortiert sind.
Testen Sie Ihre Implementierung mit dem Automaten aus Aufgabe 3.1 im Anhang.

Aufgabe 2 (optimierte Teilmengenkonstruktion - 1 Person)

Implementieren Sie die optimierte Teilmengenkonstruktion für NEAS mit zwei Symbolen (a und b) in OCaml. Als Eingabe soll Ihr Programm eine Zustands-Übergangstabelle erwarten. Diese wird als Liste von 4-Tupeln übergeben, wobei das erste Element der Zustandstyp, das zweite der Quell-Zustand und die letzten zwei Elemente des 4-Tupels die Listen der Nummern der Ziel-Zustände sind. Der Zustandstyp ist in den allgemeinen Definitionen festgeschrieben. Ist für einen Zustand des NEA ein Übergang mit einem Symbol nicht definiert, so ist der Zielzustand die leere Liste.
Die Definition der Eingabe ist in OCaml also

```
type nfa_transition = state_type * int * int list * int list;;

type nfa_transition_table = nfa_transition list;;
```

Als Ausgabe wird die Zustandsübergangstabelle des generierten DEA erwartet. Diese soll eine Liste von 4-Tupeln sein. Das erste Element ist der Typ des Zustands. Die restlichen drei Elemente sind Listen von Zuständen.
Die Definition in OCaml ist also

```
type dfa_transition = state_type * int list * int list * int list;;

type dfa_transition_table = dfa_transition list;;
```

Stellen Sie sicher, dass die Listen von Zuständen jeden Zustand nur einmal enthalten und die Elemente der Listen aufsteigend sortiert sind. Weiterhin dürfen Sie nicht die nicht optimierte Teilmengenkonstruktion so anpassen, dass die nicht erreichbaren Zustände am Ende eliminiert werden. Ermitteln Sie also zuerst die erreichbaren Zustände.
Testen Sie Ihre Implementierung mit dem Automaten aus Aufgabe 3.1 im Anhang.

Aufgabe 3 (Minimierung von DEAs - 1 Person)

Implementieren Sie den Table-Filling-Algorithmus (mitsamt Vorverarbeitung) für einen gegebenen DEA mit zwei Symbolen. Für einen gegebenen DEA mit folgender Definition in OCaml

```
type dfa_transition = state_type * int * int * int;;

type dfa_transition_table = dfa_transition list;;
```

müssen Sie also nicht erreichbare Zustände eliminieren und dann das Table-Filling ausführen.
Als Ausgabe wird der minimierte DEA erwartet. Dieser wird in OCaml wie folgt definiert

```
type min_dfa_transition = state_type * int list * int list * int list;;

type min_dfa_transition_table = min_dfa_transition list;;
```

Testen Sie Ihre Implementierung mit δ_1 und δ_2 aus Aufgabe 3.2 im Anhang.

Aufgabe 4 (Äquivalenz von DEAs - 2 Personen)

Implementieren Sie die Prüfung von DEAs mit zwei Symbolen auf Äquivalenz mithilfe des Table-Filling-Algorithmus. Implementieren Sie dazu die Minimierung von DEAs sowie die Vereinigung zweier Übergangstabellen. Als Eingabe erhalten Sie ein Tupel von Übergangstabellen. Als Ausgabe wird ein Tupel aus einem Booleschen Wert und einer Übergangstabelle erwartet. Die Definition in OCaml ist hierbei

```
type dfa_transition = state_type * int * int * int;;

type dfa_transition_table = nfa_transition list;;

type candidates = dfa_transition_table * dfa_transition_table;;

type min_dfa_transition = state_type * int list * int list * int list;;

type min_dfa_transition_table = min_dfa_transition list;;

type equivalence_result = bool * min_dfa_transition_table;;
```

Testen Sie Ihre Implementierung mit den Automaten aus Aufgabe 3.2 im Anhang.

Aufgabe 5 (CNF-Transformation - 2 Personen)

Implementieren Sie die Transformation einer beliebigen kontextfreien Grammatik in Chomsky-Normalform. Diese Aufgabe wird von zwei Personen bearbeitet, die unabhängig voneinander arbeiten können.
Die erste Person

1. eliminiert alle Epsilon-Produktionen
2. eliminiert alle Einheitsproduktionen
3. eliminiert alle unnützen Symbole

Die zweite Person

1. Separiert Terminal-Symbole und Nicht-Terminalsymbole
2. Spaltet Produktionen auf

Als Eingabe erhalten Sie die Definition der Grammatik in OCaml.
Der Typ nonterminal definiert die uns bekannten Nichtterminale, die lediglich chars sind. Der Typ terminal definiert die uns bekannten terminale, die ebenfalls chars sind.
Der Typ symbol vereinigt die Typen terminal, nonterminal und Epsilon, damit wir die rechten Seiten der Produktionen sauber abbilden können.
Eine Produktion ist als Tupel aus Nichtterminal (linke Seite) und einer Liste von Symbolen (rechte Seite).
Der Typ grammar definiert das uns bekannte Grammatik-Tupel, das sowohl die Eingabe, als auch die Ausgabe ist.

```
type nonterminal = Nonterminal of char;;
type terminal    = Terminal of char;;

type symbol      = Eps |
                   Nonterminal of char |
                   Terminal of char;;

type production = nonterminal * symbol list;;

type grammar    = nonterminal list * terminal list *
                  production list * nonterminal;;
```

Beide Personen müssen sicherstellen, dass:

- die Schnittmenge zwischen den Mengen von Terminalen und Nichtterminalen leer ist.
- die Liste der Produktionen nicht leer ist.

Person 2 muss weiterhin sicherstellen, dass

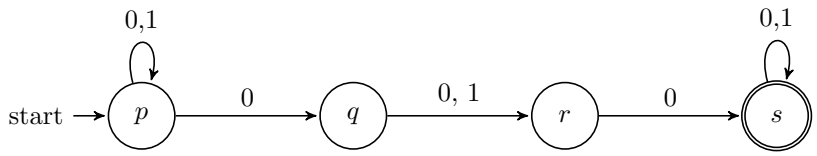
- keine der Produktionen auf Eps abbildet.
- keine Einheitsproduktionen vorhanden sind.
- alle Produktionen erreichbar sind.
- für alle Nichtterminale eine erzeugende Produktion existiert.

Wird ein Fehler in der Eingabe festgestellt, so muss die Verabeitung der Grammatik mit einer sinnvollen (aussagekräftigen) Fehlermeldung beendet werden.
Testen Sie Ihre Implementierung mit der Grammatik aus Aufgabe 3.3 im Anhang.

3 Anhang

3.1 Teilmengenkonstruktion (ohne ε-Übergänge)

Bestimmen Sie für den NEA $N = (\{p, q, r, s\}, \{0, 1\}, \delta, p, \{s\})$ mit unten abgebildeter Zustandsüberführungsfunktion den aus der Teilmengenkonstruktion hervorgehenden DEA. Bestimmen Sie weiterhin, welche der Zustände des DEAs erreicht werden.



Lösung: Die Teilmengenkonstruktion ergibt folgenden DEA

$$A_1 = (\mathcal{P}(\{p,q,r,s\}), \{0,1\}, \delta', \{p\}, \mathcal{P}(\{p,q,r,s\}) \backslash \mathcal{P}(\{p,q,r\}),$$

wobei δ' durch folgende Übergangstabelle gegeben ist. Die erreichbaren Zustände sind in der Reihenfolge ihres Auftretens indiziert worden.

δ'	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{p\}_1$	$\{p,q\}_2$	$\{p\}$
$\{q\}$	$\{r\}$	$\{r\}$
$\{r\}$	$\{s\}$	$\emptyset$
$*\{s\}$	$\{s\}$	$\{s\}$
$\{p,q\}_2$	$\{p,q,r\}_3$	$\{p,r\}_4$
$\{p,r\}_4$	$\{p,q,s\}_6$	$\{p\}$
$*\{p,s\}_8$	$\{p,q,s\}$	$\{p,s\}$
$\{q,r\}$	$\{r,s\}$	$\{r\}$
$*\{q,s\}$	$\{r,s\}$	$\{r,s\}$
$*\{r,s\}$	$\{s\}$	$\{s\}$
$\{p,q,r\}_3$	$\{p,q,r,s\}_5$	$\{p,r\}$
$*\{p,q,s\}_6$	$\{p,q,r,s\}$	$\{p,r,s\}$
$*\{p,r,s\}_7$	$\{p,q,s\}$	$\{p,s\}_8$
$*\{q,r,s\}$	$\{r,s\}$	$\{r,s\}$
$*\{p,q,r,s\}_5$	$\{p,q,r,s\}$	$\{p,r,s\}_7$

Die optimierte Teilmengenkonstruktion hingegen ergibt folgenden DEA

$$A_2 = (\{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7\}, \{0,1\}, \delta'', \{q_0\}, \{q_3,q_5,q_6,q_7\}),$$

wobei δ'' durch folgende Übergangstabelle gegeben ist.

δ''	0	1
$\rightarrow \{p\} = q_0$	$\{p,q\}$	$\{p\}$
$\{p,q\} = q_1$	$\{p,q,r\}$	$\{p,r\}$
$\{p,r\} = q_2$	$\{p,q,s\}$	$\{p\}$
$*\{p,s\} = q_3$	$\{p,q,s\}$	$\{p,s\}$
$\{p,q,r\} = q_4$	$\{p,q,r,s\}$	$\{p,r\}$
$*\{p,q,s\} = q_5$	$\{p,q,r,s\}$	$\{p,r,s\}$
$*\{p,r,s\} = q_6$	$\{p,q,s\}$	$\{p,s\}$
$*\{p,q,r,s\} = q_7$	$\{p,q,r,s\}$	$\{p,r,s\}$

3.2 Table-Filling-Algorithmus

Gegeben seien die beiden endlichen Automaten

$$A_1 := (\{A,B,C,D\}, \{0,1\}, \delta_1, A, \{C,D\}) \text{ und}$$

$$A_2 := (\{E,F,G,H,I,K\}, \{0,1\}, \delta_2, E, \{I,K\}),$$

wobei δ_1 und δ_2 durch die folgenden Überführungstabellen definiert sind:

δ_1	0	1
$\rightarrow A$	A	B
B	C	D
$*C$	C	B
$*D$	C	B

δ_2	0	1
$\rightarrow E$	F	G
F	E	H
G	I	I
H	I	I
$*I$	I	H
$*K$	K	I

1. Beweisen Sie $L(A_1) = L(A_2)$.
2. Geben Sie anschließend einen zu A_1 äquivalenten minimalen DEA an.
3. Geben Sie nun einen zu A_2 äquivalenten minimalen DEA an.

Lösung:

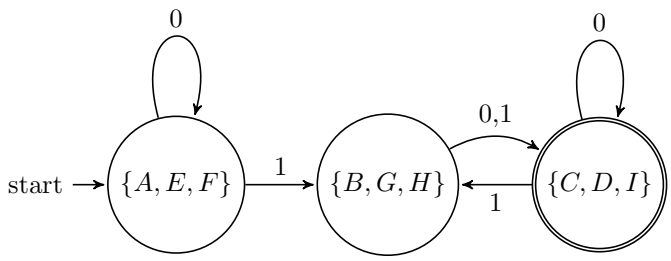
1. Um die geforderte Gleichheit zu zeigen, müssen A_1 und A_2 in einen gemeinsamen DEA "zusammengepackt" und die Äquivalenz der alten Startzustände – der neue Startzustand ist einer der beiden alten – in diesem gemeinsamen Automaten geprüft werden.
   - Der Zustand K ist nicht erreichbar und kann eliminiert werden
   - Wir wenden den Table-Filling-Algorithmus auf die Restzustände an und erhalten so folgende Tabelle:

I	X_0	X_0			X_0	X_0	X_0	X_0
H	X_1		X_0	X_0	X_1	X_1		
G	X_1		X_0	X_0	X_1	X_1		
F		X_1	X_0	X_0				
E		X_1	X_0	X_0				
D	X_0	X_0						
C	X_0	X_0						
B	X_1							
	A	B	C	D	E	F	G	H

     Erläuterung: Die Tabelle wurde spaltenweise (links beginnend) von unten nach oben berechnet. Mit X_0 werden Endzustände von Nichtendzuständen unterschieden. In jeder weiteren Runde werden in der Runde berechnete Unterschiede bereits im Verlauf derselben Runde benutzt. Unterschiede erhält man dabei durch Übergänge zu bereits unterscheidbaren Zuständen. Bspw. kann A von B unterschieden werden, weil 0 von A bzw. B nach A bzw. C führt, zwei bereits als unterscheidbar gekennzeichnete Zustände. In der Runde 2 kommen keine weiteren Zustände hinzu. Die Tabelle ist vollständig berechnet. Wir erhalten als Ergebnis die Äquivalenzklassen $\{A, E, F\}$, $\{B, G, H\}$ und $\{C, D, I\}$.
   - Da die beiden Startzustände A und E in derselben Äquivalenzklasse liegen, sind die Automaten A_1 und A_2 äquivalent, d.h. $L(A_1) = L(A_2)$.
2. Wir erhalten

   $$A_{\min} = (\{\{A, E, F\}, \{B, G, H\}, \{C, D, I\}\}, \{0, 1\}, \delta, \{A, E, F\}, \{\{C, D, I\}\}),$$

   wobei δ durch die unten abgebildete Graphik gegeben ist:



3.3 Umwandlung in CNF

Sei $G = (\{S, A, B, C\}, \{0, 1\}, P, S)$ eine Grammatik, wobei P durch folgende Menge beschrieben wird:

$$\begin{aligned} P = \{&S \to ASA \,|\, 1B \,|\, B, \\ &A \to B \,|\, S \,|\, 1C, \\ &B \to 0 \,|\, \varepsilon, \\ &C \to 1C|0CA\} \end{aligned}$$

Transformieren Sie die Grammatik in eine Chomsky-Normalform. Gehen Sie dabei nach dem in der Vorlesung beschriebenen Verfahren vor und geben Sie alle Zwischenschritte an.

Lösung:

1. Entfernung der ε-Produktionen:

   Eliminierbare Symbole sind B, A und S. Entfernung der ε-Produktionen liefert die Grammatik $G_2 = (\{S, A, B, C\}, \{0, 1\}, P_2, S)$,
   wobei P_2 gegeben ist durch

$$\begin{aligned} P_2 = \{&S \to ASA \,|\, AS \,|\, AA \,|\, SA \,|\, S \,|\, A \,|\, 1B \,|\, B \,|\, 1, \\ &A \to B \,|\, S \,|\, 1C, \\ &B \to 0, \\ &C \to 1C \,|\, 0CA \,|\, 0C\} \end{aligned}$$

2. Eliminierung von Einheitsproduktionen:

   Einheitspaare sind: (S, S), (A, A), (B, B), (C, C) (S, A), (S, B), (A, B), (A, S)

   Neue Grammatik $G_3 = (\{S, A, B, C\}, \{0, 1\}, P_3, S)$, wobei P_3 gegeben ist mit

$$\begin{aligned} P_3 = \{&S \to ASA \,|\, AS \,|\, AA \,|\, SA \,|\, 1B \,|\, 1C \,|\, 1 \,|\, 0, \\ &A \to ASA \,|\, AS \,|\, AA \,|\, SA \,|\, 1B \,|\, 1C \,|\, 1 \,|\, 0, \\ &B \to 0, \\ &C \to 1C \,|\, 0CA \,|\, 0C\} \end{aligned}$$

3. Elimination unnützer Symbole

   erzeugende Symbole: 0, 1, B, A, S. Daher nur C nicht erzeugend. Nach der Elimination von C erreichbare Symbole: S, A, B, 0, 1. Also sind keine weiteren Eliminationen durchzuführen.

   Wir erhalten $G_4 = (\{S, A, B\}, \{0, 1\}, P_4, S)$, wobei P_4 gegeben ist mit

$$\begin{aligned} P_4 = \{&S \to ASA \,|\, AS \,|\, AA \,|\, SA \,|\, 1B \,|\, 1 \,|\, 0, \\ &A \to ASA \,|\, AS \,|\, AA \,|\, SA \,|\, 1B \,|\, 1 \,|\, 0, \\ &B \to 0\} \end{aligned}$$

4. Separation von Terminalsymbolen und Variablen:

   Führe neue Variable X_a und Produktion $X_a \to a$ für jedes Terminalsymbol $a \in \Sigma$ ein, das in einer Produktion $A \to \alpha$ mit $|\alpha| \geq 2$ vorkommt. Dies liefert die neue Grammatik $G_5 = (\{S, A, B, X_1\}, \{0, 1\}, P_5, S)$,

wobei P_5 gegeben ist mit

$$\begin{aligned} P_5 = \{&S \to ASA \mid AS \mid AA \mid SA \mid X_1B \mid 1 \mid 0, \\ &A \to ASA \mid AS \mid AA \mid SA \mid X_1B \mid 1 \mid 0, \\ &B \to 0, \\ &X_1 \to 1\} \end{aligned}$$

5. Aufspaltung von Produktionen $T \to \alpha$ mit $|\alpha| > 2$:
   Dies liefert die neue Grammatik $G_6 = (\{S, A, B, X_1, Y_1, Y_2\}, \{0, 1\}, P_6, S)$, wobei P_6 gegeben ist mit

$$\begin{aligned} P_6 = \{&S \to AY_1 \mid AS \mid AA \mid SA \mid X_1B \mid 1 \mid 0, \\ &A \to AY_2 \mid AS \mid AA \mid SA \mid X_1B \mid 1 \mid 0, \\ &Y_1 \to SA, \\ &Y_2 \to SA, \\ &B \to 0, \\ &X_1 \to 1\}. \end{aligned}$$

   G_6 ist die gewünschte Grammatik in CNF.