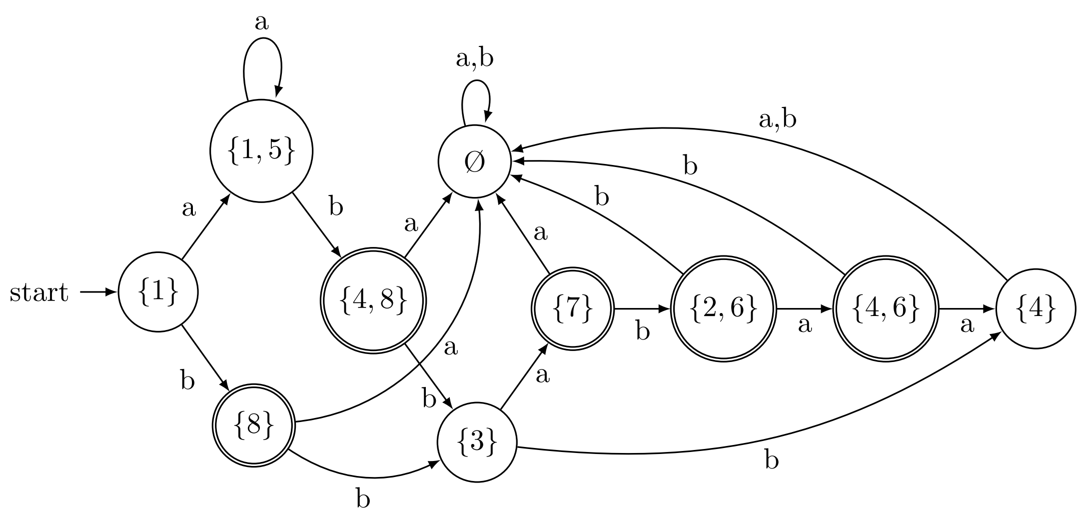
Formale Systeme, Automaten, Prozesse
Übungsblatt 4
Tutorium 11

Tim Luther, 410886
Til Mohr, 405959
Simon Michau, 406133

Aufgabe H10



Erstelle Zustandspaartabelle und streiche alle Paare, bei denen der eine Zustand Endzustand ist und der andere nicht, da diese Paare eindeutig unterscheidbar sind. Die Zellen unter der der Diagonalen ignorieren wir, da sie symmetrisch sind.

	{1}	{1,5}	{2,6}	{3}	{4}	{4,6}	{4,8}	{7}	{8}	∅
{1}	·		X			X	X	X	X	
{1,5}		·	X			X	X	X	X	
{2,6}			·	X	X					X
{3}				·		X	X	X	X	
{4}					·	X	X	X	X	
{4,6}						·				X
{4,8}							·			X
{7}								·		X
{8}									·	X
∅										·

Erstelle nun Übergangstabelle mit allen Paaren die noch nicht als unterscheidbar bekannt sind und streiche alle Zustandspaare die für a oder b auf einem unterscheidbaren Zustand landen.

Z_1	Z_2	a		b	
({1},	{1,5})	({1,5},	{1,5})	({8},	{4,8})
({1},	{3})	({1,5},	{7})		
({1},	{4})	({1,5},	{∅})	({8},	{∅})
({1},	{∅})	({1,5},	{∅})	({8},	{∅})
({1,5},	{3})	({1,5},	{7})		
({1,5},	{4})	({1,5},	{∅})	({4,8},	{∅})
({1,5},	{∅})	({1,5},	{∅})	({4,8},	{∅})
({2,6},	{4,6})	({4,6},	{4})		
({2,6},	{4,8})	({4,6},	{∅})		
({2,6},	{7})	({4,6},	{∅})		
({2,6},	{8})	({4,6},	{∅})		
({3},	{4})	({7},	{∅})		
({3},	{∅})	({7},	{∅})		
({4},	{∅})	({∅},	{∅})	({∅},	{∅})
({4,6},	{4,8})	({4},	{∅})	({∅},	{3})
({4,6},	{7})	({4},	{∅})	({∅},	{2,6})
({4,6},	{8})	({4},	{∅})	({∅},	{3})
({4,8},	{7})	({∅},	{∅})	({3},	{2,6})
({4,8},	{8})	({∅},	{∅})	({3},	{3})
({7},	{8})	({∅},	{∅})	({2,6},	{3})

Somit sind die ununterscheidbaren Zustandspaare:

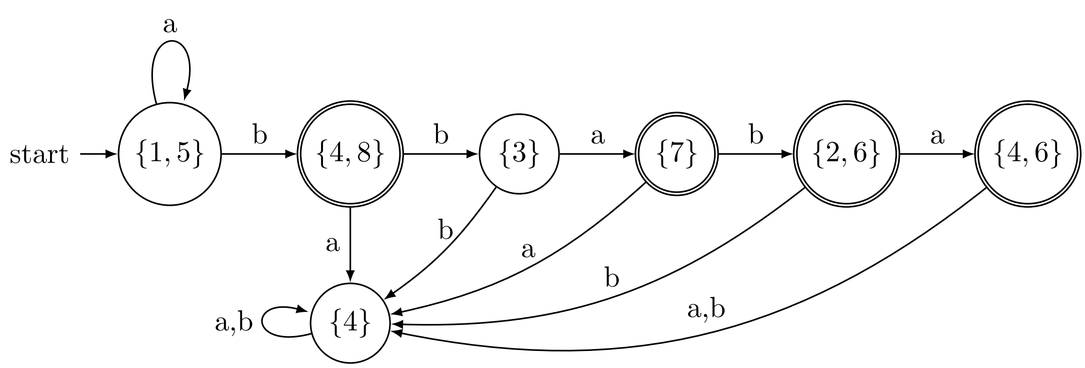
Z_1	Z_2	a		b	
({1},	{1,5})	({1,5},	{1,5})	({8},	{4,8})
({4},	{∅})	({∅},	{∅})	({∅},	{∅})
({4,8},	{8})	({∅},	{∅})	({3},	{3})

und es ergibt sich die finale Zustandspaartabelle, in der o für ein ununterscheidbares Zustandspaar steht:

Tim Luther, 410886
Til Mohr, 405959
Simon Michau, 406133

	{1}	{1,5}	{2,6}	{3}	{4}	{4,6}	{4,8}	{7}	{8}	∅
{1}	·	o	X	X	X	X	X	X	X	X
{1,5}		·	X	X	X	X	X	X	X	X
{2,6}			·	X	X	X	X	X	X	X
{3}				·	X	X	X	X	X	X
{4}					·	X	X	X	X	o
{4,6}						·	X	X	X	X
{4,8}							·	X	o	X
{7}								·	X	X
{8}									·	X
∅										·

Erstelle nun minimalen deterministischen Automaten (mit {{1},{1,5}}={1,5}, {{8},{4,8}}={4,8} und {{4},{∅}}={4}):



Aufgabe H11

a) $L_1 = \{a^n | \sqrt{n} \in \mathbb{N}, n > 100\} = \{a^{121}, a^{144}, a^{169}, ...\} = \{a^{m^2} | m \in \mathbb{N}, m > 10\}$
Sei $m \in \mathbb{N}, m > 10$ gegeben. Dann ist $w \coloneqq a^{m^2} \in L_1$. Dann kann man w zu xyz zerlegen, mit $|xy| \leq m, y \neq \epsilon$.

$$m^2 = |xyz| < |xy^2z| = |xyz| + |y| \leq m^2 + m$$

Jedoch ist das nach w in L_1 nächstgrößere Wort nur

$$(m+1)^2 = m^2 + 2m + 1$$

groß.
Damit ist klar, dass $xy^2z \notin L_1$ ist. Dadurch gilt das Pumping-Lemma nicht, weshalb L_1 keine reguläre Sprache ist.

Tim Luther, 410886
Til Mohr, 405959
Simon Michau, 406133

b)

Aufgabe H12

Regex Matcher			
Language	GoLang	Java	Java
Algorithm	NFA	NFA	backtracking
1	$79ms$	$139ms$	$8ms$
2	$132ms$	$116ms$	$4ms$
3	$153ms$	$145ms$	$8ms$
4	$21ms$	$124ms$	$15ms$
5	$23ms$	$162ms$	$30ms$
6	$101ms$	$128ms$	$60ms$
7	$81ms$	$136ms$	$90ms$
8	$29ms$	$173ms$	$49ms$
9	$31ms$	$164ms$	$96ms$
10	$54ms$	$169ms$	$171ms$
11	$35ms$	$146ms$	$197ms$
12	$35ms$	$151ms$	$373ms$
13	$56ms$	$362ms$	$668ms$
14	$40ms$	$179ms$	$1253ms$
15	$42ms$	$147ms$	$2109ms$
16	$43ms$	$193ms$	$3648ms$
17	$88ms$	$121ms$	$6160ms$
18	$95ms$	$120ms$	$10304ms$
19	$76ms$	$110ms$	$16812ms$
20	$163ms$	$72ms$	$27152ms$
Average Time	$68ms$	$152ms$	$3460ms$

a) Man kann erkennen, dass GoLang, welce NFAs verwendet, in etwa für alle $i \in \{1...20\}$ in etwa gleich schnell arbeitet. Die Ausreißer lassen sich beisielsweise durch Unterbrechung des Programms durch andere Programm erklären.
Java hingegen benutzt backtracking. Er versucht also jedes Zeichen des Inputs mit dem Regex zu matchen. Falls dies nicht geht, wird Java also die letzten durchläufe zurückgehen und einen anderen Weg einschlagen. Für kleine Eingaben ($i \in \{1...5\}$) ist dies sehr schnell, aber mit wachsendem Input wird es exponentiell aufwendiger.

b) Die Unterschiede in der Laufzeit kommen daher, dass beide Sprachen verschiedene Ansätze haben, um herauszufinden, ob ein String einem

Tim Luther, 410886
Til Mohr, 405959
Simon Michau, 406133

Regex matcht. Wie in a) beschrieben, benutzt golang NFA und ist daher für alle Eingaben hier relativ schnell. Java benutzt standartmäßig backtracking, was für wachsenden Input exponentiell länger zu brauchen scheint. Backtracking versucht quasi, die Eingabe in einen Baum aufzuteilen (nach dem gegebenen Regex). Wenn ein Zweig fehlschlät, wird in einem anderen Zweig weitergearbeitet.

c) Die erste Java-Spalte in der Tabelle ist die Umsetzung des Regex mithilfe der NFA-Implementierung von letzter Woche. Damit benutzt sie wie GoLang auch einen NFA. Man stellt fest, dass hier für alle $i \in \{1...20\}$ das Programm auch etwa gleich schnell arbeitet, doch ca. $100ms$ langsamer als die GoLang Implementierung. Dies könnte verschiedene Gründe haben, beispielsweise dass GoLang direkt zu Byte-Code compiled wird, Java nicht. Eventuell kann unsere Java NFA-Implementierung auch noch optimiert werden, wodurch auch Laufzeitunterschiede zu erklären sind.