

Eine Komplexitätstheorie von Adventure-Spielen

Seminar "Perlen der Theoretischen Informatik" AG Meyer auf der Heide

WS 2004/2005

Martin Meeser, #6146074

Betreuer: Martin Ziegler

1 Einleitung

Das dieser Seminararbeit zugrunde liegende Paper „The Theory of Finite-State Adventures“ [1] betont ausdrücklich den nicht ganz ernst gemeinten Charakter der Adventure-Komplexitätstheorie. Ziel ist es vielmehr, den doch eher trockenen Stoff der theoretischen Informatik mithilfe eines anschaulichen und für Studenten motivierenden Beispiels darzustellen. In dieser Arbeit werden die in [1] skizzierten Probleme aufgenommen und die Lösungen detailliert dargestellt. Die behandelten Themen sind formale Sprachen und die Chomsky-Hierarchie, Berechenbarkeitsmodelle und deren Hierarchie, Schnittbildung, Pumping-Lemma für reguläre und kontextfreie Sprachen, das Leerheitsproblem, Petri-Netze und das Überdeckungsproblem, das Halteproblem und schließlich Reduktionen.

2 Berechenbarkeitsmodelle und Formale Sprachen

2.1 Chomsky-Hierarchie und Hierarchie der Berechenbarkeitsmodelle

Grammatiken

Eine Grammatik G ist beschrieben durch ein Vier-Tupel $G = (V, \Sigma, P, S)$, wobei es sich bei V um die Variablen, bei Σ um die Terminale, bei P um die Produktionen und bei S um das Startsymbol handelt.

Turing-Maschinen, Chomsky-0-Grammatiken (rekursiv aufzählbare Grammatiken)

Turing-Maschinen(TM) stellen ein adäquates und allgemeines Modell zur mathematischen Darstellung von Computern dar.

Die Chomsky-0-Grammatiken sind genau diejenigen Sprachen, die von Turing-Maschinen erkannt werden. Für die Produktionen muss gelten, dass die linke Seite mindestens eine Variable enthält ($P \subseteq ((V \cup \Sigma)^+ - \Sigma) \times (V \times \Sigma)^*$).

Bei Turing-Maschinen steht zu Beginn einer Berechnung nicht fest, ob die Berechnung enden wird. Dies ist nicht der Fall, wenn die Turing-Maschine in eine Endlosschleife gerät.

Linear beschränkte Turing-Maschinen, Chomsky-1-Grammatiken (kontextsensitive Grammatiken)

Linear beschränkte Turing-Maschinen(LBTM) verhalten sich wie normale Turing-Maschinen, allerdings ist ihr Speicherplatz begrenzt und zwar in einer linearen Größenordnung abhängig von der Eingabegröße. Dadurch ist sichergestellt, dass LBTMs immer halten.

Die Chomsky-1-Grammatiken sind genau diejenigen Grammatiken, die durch LBTMs erkannt werden.

Kontextsensitive Grammatiken machen eine erste Einschränkung bezüglich der Produktionsregeln ($\forall (u \rightarrow v) \in P: |u| \leq |v|$).

Stapelautomaten, Chomsky-2-Grammatiken (kontextfreie Grammatiken)

Ein Stapelautomat (PDA) kann die Eingabe nur einmal durchlaufen. Er hat die Fähigkeiten, ein Symbol oben auf dem Stapel abzulegen (speichern) oder oben vom Stapel zu nehmen (lesen).

Das Pumping Lemma für kontextfreie Sprachen besagt, dass es innerhalb eines kontextfreien Wortes immer zwei kurze Teilwörter gibt, die beide gleich aber beliebig oft wiederholt werden dürfen.

Die kontextfreien Sprachen sind genau diejenigen Sprachen, die von einem nichtdeterministischen Kellerautomaten erkannt werden. Bei kontextfreien Grammatiken handelt es sich um Chomsky-1 Grammatiken mit der weiteren Einschränkung, dass umgangssprachlich „auf der linken Seite nur Variablen stehen dürfen“ ($\forall(u \rightarrow v) \in P: u \in V$).

Der Schnitt einer kontextfreien Sprache mit einer regulären Sprache ist wieder kontextfrei. Es gibt kontextfreie Sprachen, deren Schnitt nicht kontextfrei ist.

Endliche Automaten, Chomsky -3-Grammatiken (reguläre Grammatiken)

Charakteristisch für endliche Automaten ist, dass sie die Eingabe nur einmal durchlaufen können. Daher fasst ein Zustand des Systems alle Informationen betreffend vorhergehender Eingaben zusammen, um das Verhalten des Systems bei nachfolgenden Eingaben bestimmen zu können. Das Pumping Lemma beschreibt eine fundamentale Eigenschaft der endlichen Automaten. Gegeben sei ein Automat mit n Zuständen. Sei w ein Wort mit $|w| > n$. Dann muss der Automat zwangsläufig einige Zustände mehr als einmal benutzt haben, es gibt innerhalb des Automaten eine Schleife, die beliebig oft durchlaufen werden kann.

Die regulären Sprachen sind genau diejenigen Sprachen, die von einem endlichen Automaten erkannt werden, sie sind noch eingeschränkter als die Chomsky-2-Grammatiken ($\forall(u \rightarrow v) \in P: v = \varepsilon, v = a, v = aw \quad a \in \Sigma; w \in V$).

Der Schnitt zweier regulärer Sprachen ist wieder eine reguläre Sprache.

Determinismus und Nichtdeterminismus

Die nichtdeterministischen Modelle unterscheiden sich von den deterministischen dadurch, dass der Automat Wahlmöglichkeiten besitzt. Man geht davon aus, dass der Automat in Laufzeit $O(1)$ die richtige Wahlmöglichkeit auswählt, falls sie existiert.

Die Idee des Nichtdeterminismus ist rein mathematisch und lässt sich in der Praxis nicht realisieren. Bei Turing-Maschinen und endlichen Automaten lassen sich die nichtdeterministischen Modelle zu deterministischen transformieren (mit entsprechendem Aufwand). Bei Kellerautomaten existiert ein echter Unterschied zwischen den beiden Modellen.

Grammatik	Berechnungsmodell	Beispiel	Chomsky-Hierarchie
nicht rekursiv aufzählbar		DIAG, \bar{H}	
rekursiv aufzählbar	DTM, NTM akzeptiert	H	Chomsky-0
rekursiv (entscheidbar)	DTM, NTM entscheidet	Band-hierarchiesatz	
kontextsensitiv	NLBA akzeptiert (NSPACE(n))	$(a^n b^n c^n)$ (ww)	Chomsky-1
kontextfrei (nicht-deterministisch kontextfrei)	NPDA akzeptiert	$(0^n 1^n)$ (ww ^R)	Chomsky-2
deterministisch kontextfrei	DPDA akzeptiert	$(w\$w^R)$	
regulär	DFA, NFA akzeptiert	$(0^n 1)$	Chomsky-3

Abbildung 1: Übersicht über die Berechnungsmodelle nach [2].

2.2 GOTO - Programme

GOTO - Programme (und auch WHILE - Programme) sind bezüglich ihrer Berechenbarkeits-Möglichkeiten äquivalent zu Turing-Maschinen.

Ein GOTO - Programm kann durch die folgende Syntax beschrieben werden:

$$P ::= M:A;P \mid M: HALT$$

Dabei handelt es sich bei M um Marken (für Sprünge) und bei A um eine Anweisungen der Form:

$$A ::= x_i := 0 \mid x_i := x_i + 1 \mid x_i := x_i - 2 \mid GOTO M \mid IF (x_i = 0) THEN GOTO L \mid HALT$$

Die Variablen können nur positive Werte annehmen, entsprechend ist $0 - 1 = 0$ definiert. Initial sind alle Variablen auf 0 gesetzt.

2.3 Petri-Netze

Zunächst muss betont werden, dass es sich nicht um ein Berechnungsmodell handelt, sondern um eine Modellierungsmöglichkeit von Abläufen mit nebenläufigen Prozessen und kausalen Beziehungen.

Definition Petri-Netze

Ein Petri-Netz ist ein Tripel $P = (S, T, F)$ mit $S \cap T = \emptyset$. S ist eine Menge von Stellen und repräsentiert Bedingungen oder Zustände. Die Stellen werden graphisch als Kreise dargestellt. Bei T handelt es sich um eine Menge von Transitionen, die Aktivitäten repräsentieren und graphisch als Rechtecke dargestellt werden. F beschreibt Relationen mit $F \subseteq S \times T \cup T \times S$. P bildet einen bipartiten, gerichteten Graphen mit Knoten $S \cup T$ und den Kanten F . Der Vorbereich einer Transition t ist definiert als $Vorbereich(t) := \{s \mid (s, t) \in F\}$, entsprechend $Nachbereich(t) := \{s \mid (t, s) \in F\}$.

Der Zustand des Petri-Netzes wird durch eine Markierungsfunktion angegeben, die jeder Stelle eine Anzahl von Marken zuordnet: $M: S \rightarrow N_0$. Die Marken werden als schwarze Punkte in den Stellen gezeichnet.

Ein Petri-Netz heißt *unbeschränkt*, sobald es mindestens eine Stelle gibt, die mit unendlich vielen Marken markiert werden kann. Ein Petri-Netz heißt *1-sicher*, falls alle Stellen mit nur einer oder keiner Marke markiert werden kann.

Schaltregel

Eine Transition kann schalten, wenn ihr gesamter Vorbereich markiert ist (die Transition ist aktiviert). Beim Schalten verändert die Transition die Markierung m zu einer Nachfolgemarkierung m' wie folgt: $m'(v) = m(v) - 1, v \in Vorbereich(t); m'(n) = m(n) + 1, n \in Nachbereich(t)$. Es kann immer nur eine Transition schalten. Sind mehrere Transitionen aktiviert, wird eine nichtdeterministisch ausgewählt.

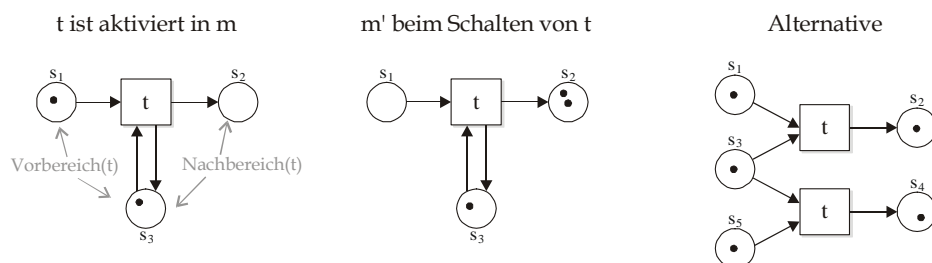


Abb. 2: kurze Beispiele zu Petrinetzen

Erreichbarkeitsgraph

Der Erreichbarkeitsgraph stellt alle von der Anfangsmarkierung aus erreichbaren Markierungen dar. Er kann systematisch konstruiert werden, indem man die Markierungen als Zustände und die entsprechenden schaltenden Transitionen als Kanten konstruiert. In beschränkten Netzen ist der Erreichbarkeitsgraph immer endlich, in unbeschränkten Netzen ist er unendlich.

Einordnung in die Hierarchie der Berechenbarkeitsmodelle

Von ihrer Komplexität her sind Petri-Netze zwischen den Turing-Maschinen und den NLBAs einzuordnen. Sie unterscheiden sich von Turing-Maschinen in zwei Punkten: Es ist entscheidbar, ob es endliche viele Markierungen (Zustände) gibt und ob eine Markierung von einer anderen aus erreichbar ist.

In 1-sicheren Netzen gibt es 2^n verschiedene Markierung, sie sind wesentlich weniger komplex.

Überdeckungsbaum

Der Überdeckungsbaum ist eine Art „erweiterter Erreichbarkeitsgraph“. Bei der Konstruktion des Überdeckungsbaumes kann festgestellt werden, ob eine Stelle unbeschränkt ist. Daher ist ein Überdeckungsbaum immer endlich.

Das Überdeckungsproblem stellt die Frage: „Wird die Stelle s jemals mit einer Markierung belegt?“. Dies ist mit Hilfe des Überdeckungsbaumes entscheidbar. Allerdings wächst seine Größe primitiv rekursiv innerhalb der Größe des Petrinetzes, d.h. die Konstruktion ist in der Regel nicht praktisch durchführbar.

3 Adventure-Spiele und grundlegende Betrachtungsweise

3.1 Adventure-Spiele





Adventure-Karten

Eine Adventure-Karte M (Map) ist definiert durch einen nichtdeterministischen endlichen Automaten.

$$AK\ M = NFA\ A = (Q, \Sigma, \delta, q_0, F)$$

Das Eingabealphabet ist festgelegt auf:

$$\Sigma = \{$$

- „d“  (Drachen),
- „s“  (Schwerter),
- „a“  (arch / Torbögen),
- „r“  (river / Flüsse),
- „g“  (gate / Türen),
- „t“  (treasure / Schätze),
- „k“  (key / Schlüssel)

}

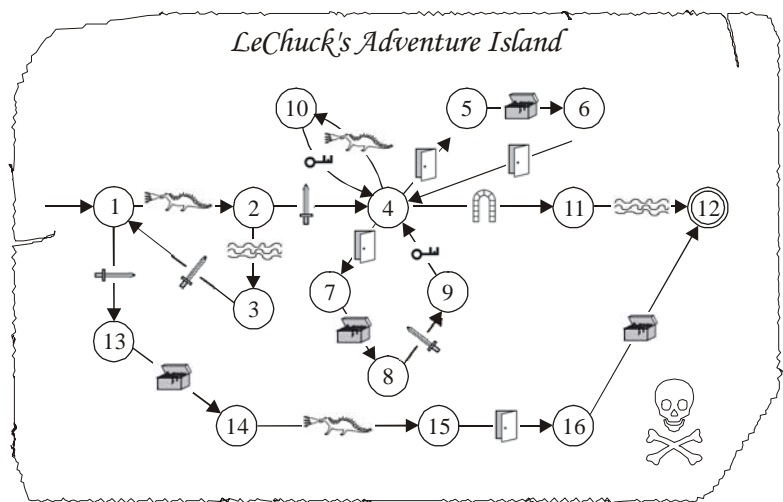


Abb. 3: Eine Beispiel Adventure-Karte.

Der Abenteurer beginnt an der Startposition / dem Startzustand

q_0 und muss einen der Endzustände erreichen.

Mit $L(M)$ bezeichnen wir diejenige Sprache, die

von dem NFA akzeptiert wird, der die Karte repräsentiert.

Adventure-Regeln

Die Adventure-Regeln (R) beschreiben Einschränkungen und Regeln unterschiedlicher Komplexität, die bei dem Passieren der Kanten berücksichtigt werden müssen. Dies bedeutet beispielsweise, dass eine Kante der Adventure-Karte nicht passiert werden darf, wenn eine gewisse Vorbedingung nicht erfüllt ist.

Mit zunehmender Adventure-Klasse gewinnen die Adventure-Regeln an Komplexität hinzu.

Adventure-Spiele

Ein Adventure-Spiel (FSA – Finite State Adventure) setzt sich zusammen aus der Adventure-Karte und den Adventure Regeln.

$$FSA\ A = (AK\ M, AR\ R)$$

Lösungen für das Adventure-Spiel sind genau all diejenigen Eingabewörter, die von dem NFA, der die Adventure-Karte repräsentiert, akzeptiert werden und die alle gegebenen Adventure-Regeln erfüllen. Wir bezeichnen diese Eingabewörter als die von dem $FSA\ A$ akzeptierte Sprache $L(A)$.

3.2 Grundlegende Betrachtungsweise

Wir interessieren uns für zwei Fragestellungen:

- In welcher Ebene der Chomsky-Hierarchie liegt Ebene i der Adventure-Hierarchie ($i=0\dots4$) bzw. durch welches Berechenbarkeitsmodell lässt sich die Adventure-Ebene i darstellen?
- Was ist die Komplexität der Lösungsfindung von Adventure-Ebene i ?

Um die erste Frage beantworten zu können, benötigen wir eine Funktion f , die ein gegebenes Adventure-Spiel in eine Grammatik bzw. in eines der bekannten Berechenbarkeitsmodelle transformiert.

Die Komplexität der Lösungsfindung bezieht sich auf die Frage, wie viel Laufzeit benötigt wird, um zu entscheiden, ob das Adventure eine Lösung besitzt.

In Vergleich zu den Berechenbarkeitsmodellen entspricht dies dem Leerheitsproblem ($L(G) = \emptyset?$).

Dies ist ein bekanntes Entscheidungsproblem und wird im [2] detailliert behandelt.

Haben wir mit Hilfe der Transformationsfunktion f eine implizite Zuordnung gefunden, so können wir die bekannten Algorithmen des Leerheitsproblems verwenden und entsprechende Rückschlüsse auf das Adventure der Ebene i machen.

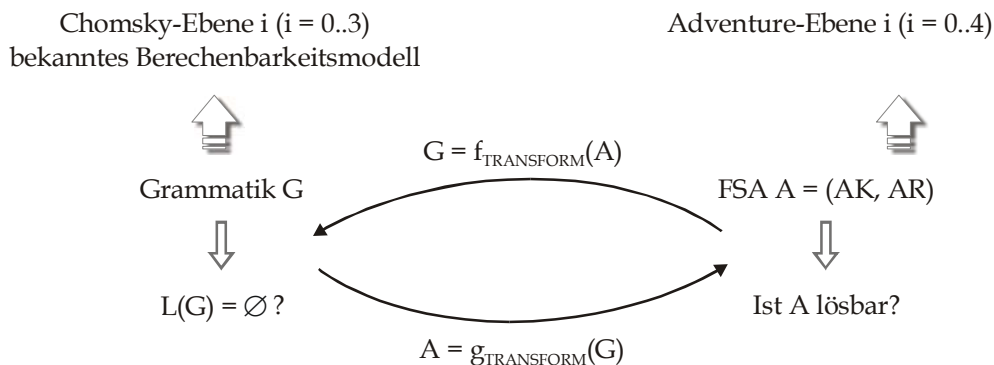


Abb. 4: Gegenüberstellung formale Sprachen und Adventure.

4 Adventure-Ebene 0 (Einen Weg Finden)

Auf der einfachsten Ebene der Adventure-Spiele beschäftigen wir uns ausschließlich mit der zugrunde liegende Karte. Für die Adventure-Regeln gilt also:

$$AR = \emptyset$$

Somit enthält $L(M)$ alle Lösungen für ein Adventure der Ebene 0 ($L(A) = L(M)$). Falls $L(M)$ leer ist, so besitzt das Adventure auch keine Lösung.

Anschaulich ist ein Adventure der Ebene 0 genau dann lösbar, wenn es einen Weg vom Startzustand zum Endzustand gibt.

Satz 4.1: Für ein Adventure der Ebene 0 lässt sich in Zeit $O(n^2)$ mit $n = |Q|$ entscheiden, ob das Adventure lösbar ist.

Beweis:

Mit einer Tiefensuche kann festgestellt werden, ob der Endzustand vom Startzustand aus erreicht werden kann.

⇒ Tiefensuche benötigt $O(|V| + |E|)$.

⇒ $E = O(|V|^2)$, da jeder der n Knoten mit maximal n weiteren Knoten verbunden sein kann.

⇒ In einer Adventure-Karte bzw. in einem endlichen Automaten entspricht V den Zuständen Q und E entspricht den Zustandsübergängen δ .

Anmerkung:

Dies entspricht dem Vorgehen, um das Leerheitsproblem bei regulären Sprachen zu lösen.

Satz 4.2: Für alle Adventures der Ebene 0 mit FSA $A = (AK, M, \emptyset)$ gilt: $L(A) \in REG$.

Beweis: Folgt direkt aus der Definition, $f_{TRANSFORM}(A)$ gibt 1 zu 1 die gegebene Adventure-Karte als Automat zurück.

Satz 4.3: Für alle $L \subseteq \Sigma^*$, $L \in REG$ gibt es ein Ebene 0 FSA mit $L = L(A)$.

Beweis: Folgt analog ebenfalls direkt aus der Definition, $g_{TRANSFORM}(G)$ gibt den NFA zurück, der die Sprache L erzeugt.

Folgerung 4.4: REG entspricht $\{ FSA = (AK, \emptyset) \}$

5 Adventure-Ebene 1 (Schätze Finden)

Die Ebene 1 führt drei erste einfache Adventure-Regeln ein. Sie lauten:

- (G) Eine Tür kann nur mit einem vorher gefundenen Schlüssel passiert werden (ein Schlüssel kann beliebig oft verwendet werden).
 Beispielhaft kann man diese Regel etwas formaler ausdrücken: In der Adventure-Karte können Tür-Kanten nur passiert werden, wenn vorher wenigstens eine Schlüssel-Kante passiert wurde.
- (D) Nachdem man einen Drachen getroffen hat, muss man anschließend sofort in einen Fluss springen, damit man nicht verbrannt wird. Hat man allerdings bereits ein Schwert gefunden, tötet man kurzerhand den Drachen und muss daher nicht in einen Fluss springen.
- (T) Es müssen mindestens zwei Schätze gefunden werden.

Alle drei Regeln haben die gemeinsame Eigenschaft, dass sie eine endliche Anzahl an Nach- bzw. Vorbedingung stellen. Daher lassen sie sich als endliche Automaten darstellen.

Es ist nun möglich, den Schnitt des NFAs, der die Adventure-Karte repräsentiert, mit den NFAs, die die Adventure-Regeln darstellen, zu bilden. All diejenigen Eingabewörter, die von dem resultierenden NFA akzeptiert werden, sind Lösungen für das Adventure der Ebene 1 ($L(A) = L(M) \cap L(G) \cap L(D) \cap L(T)$).

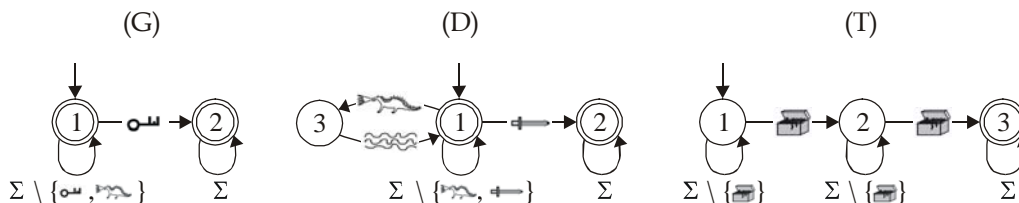


Abb. 5: Bedingungen dargestellt als endliche Automaten.

Hilfssatz 5.1: Schnittbildung von zwei NFAs ist in Laufzeit $O(n^4)$ möglich mit $n = \max(|Q_1|, |Q_2|)$.

Beweis:

NFA $A = \text{RegSchnitt}(NFA A_1, NFA A_2) \{$

1. Falls $\Sigma_1 \neq \Sigma_2$ LEHNE AB!
2. $Q = \emptyset; \delta = \emptyset; F = \emptyset$
3. Für alle $p \in Q_1$
 4. Für alle $q \in Q_2$
 5. Für alle $s \in \Sigma$
 6. $Q = Q \cup \{p, q\}$
 7. Falls $p \in F_1$ und $q \in F_2$ dann $F = F \cup \{p, q\}$
 8. Falls $\delta_1(p, s) \neq \emptyset$ und $\delta_2(q, s) \neq \emptyset$ dann $\delta[p, q, s] = \{\{\delta_1(p, s) \times \delta_2(q, s)\}\}$
 - 8a. Falls $s = \varepsilon$ dann $\delta[p, q, \varepsilon] = \{\{\delta_1(p, \varepsilon) \times \delta_2(q, \varepsilon)\}\}$
9. Gebe $A = (Q, \Sigma, \delta, [q_{(1)0}, q_{(2)0}], F)$ zurück

$\}$

Laufzeit:

- Die Operationen in 1., 2., 6. und 7. lassen sich in konstanter Laufzeit ausführen.
 - Die Schleife in 5. macht insgesamt $|\Sigma|$ Durchläufe, wobei $|\Sigma|$ konstant ist (in unserem Fall besteht Σ aus sieben Elementen).
 - In 8. muss das Kreuzprodukt der Zustandsübergänge gebildet werden. Im schlimmsten Fall ist in beiden Automaten jeder Zustand mit jedem anderen verbunden: $O(n^2)$!
 - Da $n = \max(|Q_1|, |Q_2|)$ ist, werden die Schleifen in 3. und 4. jeweils $O(n)$ mal wiederholt.
- \Rightarrow Die Laufzeit beträgt $O(n^4)$.

Anmerkung:

Betrachtet man eine Funktion $\text{RegSchnitt}(NFA A_1, DFA A_2)$, so beträgt die Laufzeit nur noch $O(n^3)$, da in 8. nur δ_1 aus einer mehr-elementigen Menge bestehen kann. Entsprechend kann man den Schnitt von zwei DFAs in Laufzeit $O(n^2)$ bilden.

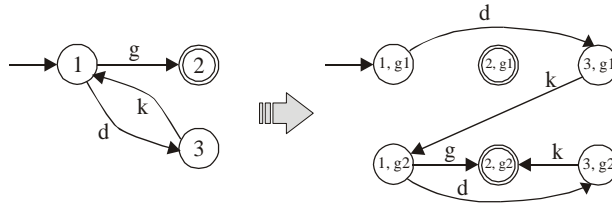


Abb. 6: Schnittbildung einer einfachen Adventure-Karte mit der Regel (G_1) .

Satz 5.2: Bei Ebene-1-Adventures lässt sich in Zeit $O(n^4)$ entscheiden, ob das Adventure lösbar ist.

Beweis:

- Zunächst wird der Schnitt der Adventure-Karte mit den drei Adventure-Regeln gebildet.
 $\Rightarrow 3 * O(n^4)$
- Anschließend können wir wie in Ebene 0 eine Tiefensuche verwenden.
 $\Rightarrow O(n^2)$
- $\Rightarrow O(n^4)$

Satz 5.3: Für alle Adventures der Ebene 1 mit FSA $A = (AK M, \{(G), (D), (T)\})$ gilt: $L(A) \in REG$.

Beweis: Folgt direkt aus der Schnittbildung wie oben beschrieben.

Folgerung 5.5: REG entspricht $\{ FSA = (AK, \{(G_1), (D_1), (T_1)\}) \}$

Anmerkung: Nach der Schnittbildung entspricht ein Ebene 1 FSA einem Adventure der Ebene 0, dass noch immer alle Regeln der Ebene 1 erfüllt.

6 Adventure-Ebene 2 (Schlüssel zählen)

Auf Ebene 2 wird die Regel (G_1) durch eine neue Regel (G_2') ersetzt. Die Regeln (D_1) und (T_1) bleiben erhalten.

(G') Ein Schlüssel verschwindet, sobald eine Tür geöffnet wurde. Die Tür schließt sich anschließend wieder. Man kann beliebig viele Schlüssel einsammeln.

Formal lässt sich diese Regel so ausdrücken: „Um n Tür-Kanten passieren zu können müssen vorher mindestens n Schlüssel-Kanten passiert werden.“

Dieses Mitzählen lässt sich mit einem endlichen Automaten nicht mehr realisieren. Es handelt sich allerdings um eine Anforderung, die von einem Kellerautomaten erfüllt wird (der Automat in Abbildung 7 kann mit vollem Keller akzeptieren). Wir können somit erneut den Schnitt zwischen der Karte und den Regeln bilden.

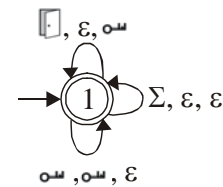


Abb. 7: (G') repräsentiert als Keller-Automat.

Satz 6.1: Diejenigen Sprachen, die einem Adventure der Ebene 2 entsprechen, sind in der Regel nicht regulär.

Es existiert mindestens ein $w \in L(A_2)$ mit FSA $A_2 = (AK, \{(D_1), (T_1), (G_2')\})$ für das gilt: $w \notin REG$.

Beweis (Pumping Lemma für reguläre Sprachen):

Die Sprache, welche die möglichen Lösungen für das Beispiel aus Abbildung 8 enthält, ist

$L = \{k^m g^n t^2 \mid m \geq n \geq 1\}$. Sei nun $x = k^p g^p t^2 \in L$ mit $p \in \mathbb{N}$ beliebig aber fest. Weiter sei $x = uv^i w = k^{p-c} (k^c)^i g^p t^2$ mit $|x| > p$ und $|uv| < p$. Mit $i = 0$ ist nun $x = k^{p-c} g^p t^2 \notin L$. Daraus folgt, dass L nicht rekursiv sein kann.

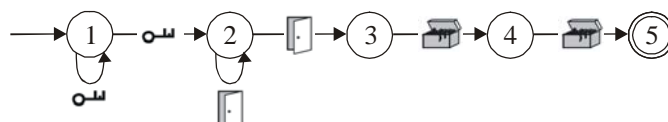


Abb. 8: Ein nicht-reguläres Adventure der Ebene 2.

Satz 6.3: Für alle FSA $A_2 = (AK M, \{(D), (T), (G')\})$ gilt: $L(A) \in CFL$.

Beweis:

Zunächst wird der Schnitt der Adventure-Karte mit den Regeln (D) und (T) gebildet. Mit der resultierenden rekursiven Sprache wird erneut der Schnitt mit dem PDA gebildet, der die Regel (G') repräsentiert. Man erhält somit eine kontextfreie Sprache $L(A) = L(M) \cap L(D) \cap L(T) \cap L(G')$.

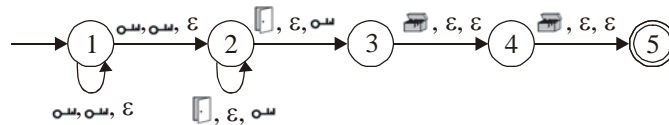


Abb. 9: Adventure aus Abb. 7 nach Schnittbildung mit (G').

Folgerung 6.5: Adventure-Ebene-0,1 \subset Adventure-Ebene-2

Folgerung 6.6: CFL entspricht $\{FSA = (Ak, \{(D), (T), (G')\})\}$

Satz (ohne Beweis): Es ist bei Ebene-2-FSAs in $O(n^3)$ entscheidbar, ob das Adventure eine Lösung besitzt, wobei n die Anzahl der Zustände in der Adventure-Karte darstellt.

Beweisskizze:

- Wir gehen davon aus, dass $L(M) \cap L(D) \cap L(T)$ als endlicher Automat vorliegt. Die Anzahl der Zustände dieses Automaten beträgt $O(n)$.
- Wir gehen ebenfalls davon aus, dass $L(G')$ als kontextfreie Grammatik vorliegt, die Anzahl der Produktionen in dieser Grammatik ist konstant.
- Wir können nun einen Algorithmus mit Laufzeit $O(p*s^3)$ verwenden, wie er in [6] beschrieben ist. Dabei beschreibt p die Anzahl der Produktionen der kontextfreien Grammatik und s die Anzahl der Zustände des endlichen Automaten.

Anmerkung: Der Platzbedarf dieses Algorithmus beläuft sich auf $O(p*s^2)$.

$\Rightarrow O(n^3)$

7 Adventure-Ebene 3 (Schlüssel und Schwerter zählen)

Die nächste Verkomplizierung macht es erforderlich, neben der Schlüsselanzahl auch die Anzahl an Schwertern mitzuzählen.

(D') Falls man ein Schwert hat und auf einen Drachen trifft, kann man den Drachen töten, allerdings wird dann das Schwert vom Drachenblut verätzt sein und kann daher anschließend nicht mehr benutzt werden. Trifft man allerdings auf einen Drachen und folgt anschließend ein Fluss, dann kann man in den Fluss springen und muss kein Schwert benutzen. Zu dem großen Bedauern aller Helden kann man Drachen nicht wirklich vernichten, direkt nach ihrer heldenhaften Tötung erwachen sie zu neuem Leben.

Dies übersteigt die Möglichkeiten eines Kellerautomaten, da man mehr als einen Keller benötigen würde um die beiden Werte mitzuzählen.

Wir sind aber in der Lage mithilfe eines Überdeckungsbaumes eines Petrinetzes zu entscheiden, ob Adventures der Ebene 3 lösbar sind.

Satz 7.1: Diejenigen Sprachen, die einem Adventure der Ebene 3 entsprechen, sind in der Regel nicht kontextfrei.

Es existiert mind. ein $w \in L(A)$ mit $FSA A = (AK, \{(T), (G'), (D')\})$ für das gilt: $w \notin CFL$.

Beweis (Pumping Lemma für kontextfreie Sprachen):

Betrachte die Karte in Abbildung 10, die mit den Ebene-3-Regeln ein Adventure ergibt, dass eine nicht kontextfreie Sprache erzeugt. Diese Sprache ist $L = \{k^k s^l g^m d^n t^2 \mid k \geq m \geq 1, l \geq n \geq 1\}$. Zur Vereinfachung lassen wir das t^2 außer Acht.

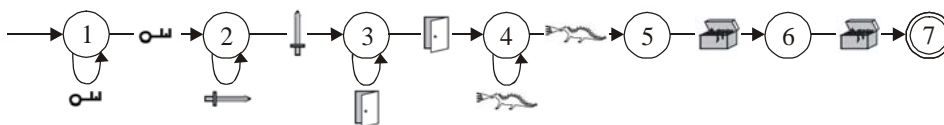


Abb. 9: Ein Adventure welches eine nicht-kontextfreie Sprache der Ebene 3 erzeugt

Sei nun $z = uvwxy = k^p s^p g^p d^p \in L$ mit $|vx| \geq 1$ und $|vwx| \leq p$ mit $p \in \mathbb{N}$ beliebig aber fest.

Mit dieser Aufteilung gibt es zunächst vier Fälle, bei denen v^iwx^i aus jeweils einem der Eingabesymbole besteht. In diesen vier Fällen ist es leicht zu sehen, dass das Wort durch auf- bzw. abpumpen nicht mehr in L liegt.

Bei den nächsten zwei Fällen besteht v^iwx^i nur aus k 's und s 's bzw. nur aus s 's und g 's. Mit $i = 0$ fehlen jeweils einige der k 's bzw. einige s 's.

Im letzten Fall ist v^iwx^i ausschließlich aus g 's und d 's zusammengesetzt. Mit $i \geq 2$ gibt es zu viele g 's und d 's.

Daher kann L nicht kontextfrei sein.

Folgerung 7.2: Adventure-Ebene 2 \subset Adventure-Ebene 3

Die Funktion, welche aus einem FSA der Ebene 3 ein Petrinetz konstruiert, hat die folgende Form:

Petrinetz $P = f_{\text{TRANSFORM}}(A, M)$ {

1. Bilde den Schnitt der Adventure-Karte M und der Schatzregel (T) .
2. Erzeuge ein dem geschnittenem Automaten entsprechendes Petrinetz:
 - a. Jeder Zustand q wird zu einer Stelle s .
 - b. Jede Kante wird zu einer entsprechenden Transition.
3. Füge eine Stelle „Schwerter“ hinzu, die die Anzahl der Schwerter mitzählt.
4. Für alle Schwerter-Transitions füge die Stelle „Schwerter“ dem Nachbereich hinzu.
5. Für alle Drachen-Transitions füge die Stelle „Schwerter“ dem Vorbereich hinzu.
6. Füge eine Stelle „Schlüssel“ hinzu, die die Anzahl der Schlüssel zählt.
7. Für alle Schlüssel-Transitions füge die Stelle „Schlüssel“ dem Nachbereich hinzu.
8. Für alle Tür-Transitions füge die Stelle „Schlüssel“ dem Vorbereich hinzu.
9. Für alle Fluss-Kanten, die direkt auf Drachen-Kanten folgen, füge eine entsprechende Drachen-Fluss-Transition dem Petri-Netz hinzu:
 - a. Die Stelle vor dem Drachen wird dem Vorbereich der Transition beigefügt.
 - b. Die Stelle nach dem Fluss wird zu dem Nachbereich hinzugefügt.
10. Setze eine Marke auf die Stelle, die den Anfangszustand repräsentiert.
11. Gebe das so konstruierte Petrinetz zurück.

}

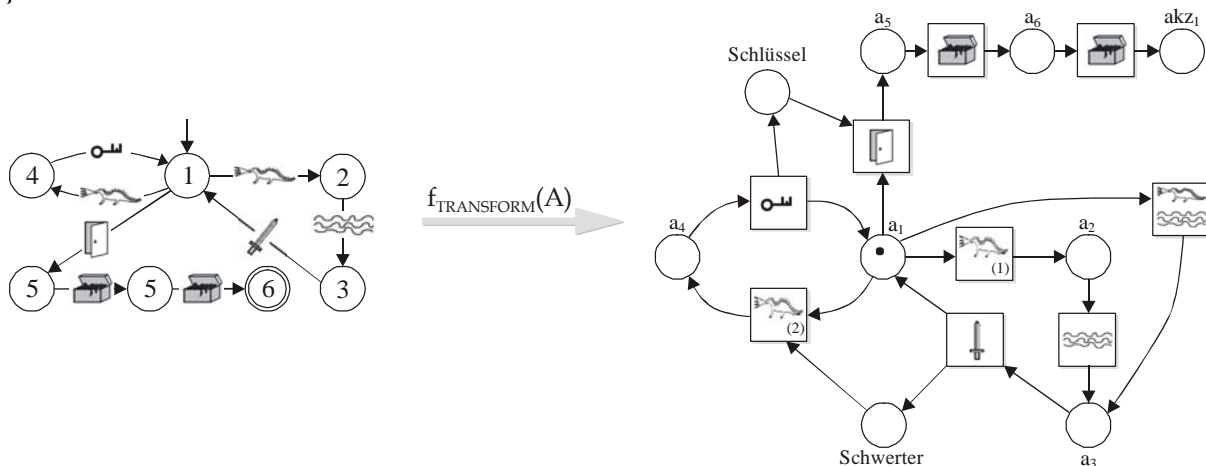


Abbildung 10: Beispielhafte Transformation eines Ebene 3 FSA in ein Petrinetz

Die Frage „Ist das Adventure der Ebene 3 lösbar?“ ist äquivalent zu dem Überdeckungsproblem von Petrinetzen. Wenn diejenigen Stellen, die die akzeptierenden Zustände der Adventure-Karte darstellen, erreichbar sind, dann besitzt das Adventure eine Lösung. Wir können daher feststellen: Es ist zwar entscheidbar, ob ein Adventure der Ebene 3 eine Lösung besitzt, es wird in der Regel aber nicht berechenbar sein, da die Größe des Überdeckungsbaumes primitiv rekursiv ist.

8 Adventure-Ebene 4 (Verbotene Flüsse und Torbögen)

Schließlich kommen in der letzten Adventure-Ebene zwei letzte Erweiterungen zu den bestehenden Regeln hinzu.

- (R) Schwerter sind viel zu schwer um mit ihnen einen Fluss zu durchqueren. Daher ist es nicht erlaubt, einen Fluss zu überqueren, falls man mindestens ein Schwert besitzt.
- (A) Die Torbögen sind magisch und lassen niemanden hindurch, der auch nur einen einzigen Schlüssel mit sich trägt.

Es ist nicht möglich, Gegenstände wegzuworfen. Die charakteristische Eigenschaft dieser Ebene ist es, dass es einen Kontrollfluss gibt, der abhängig von dem Wert von Variablen ist, d.h. in der Ebene 4 müssen Variablen auf ihren Wert geprüft werden. Konkret im unserem Fall handelt es sich dabei um die Anzahl der Schwerter und die der Schlüssel.

Satz 8.1: Das Halteproblem lässt auf das „Ist das Adventure lösbar?“ - Problem für ein Adventure der Ebene 4 reduzieren.

$$H \leq_p (L(A) = \emptyset?) \text{ mit } FSA = (Ak, \{(T), (G'), (D'), (R), (A)\})$$

Beweis:

Wir definieren zunächst die folgende Ausprägung des Halteproblems:

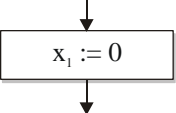
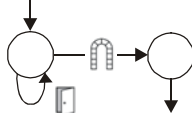
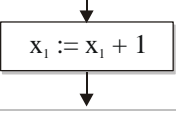
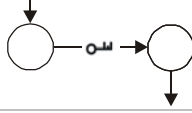
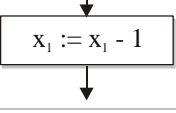
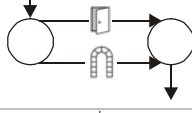
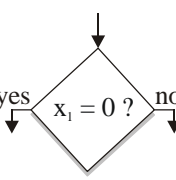
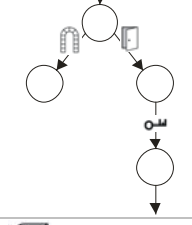

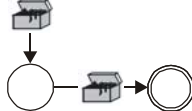
$H_{GP} = \{ \langle GP \rangle \mid GP \text{ ist eine gültige Codierung eines GOTO - Programms und } GP \text{ hält.} \}$

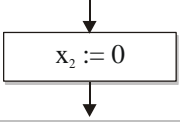
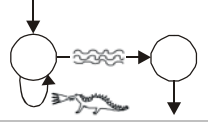
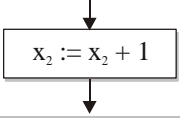
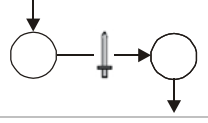
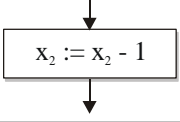
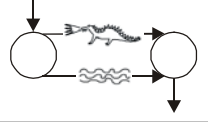
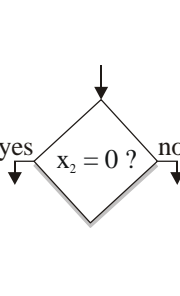
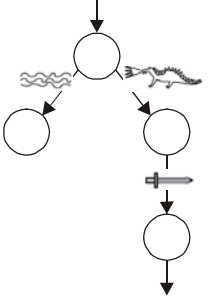
Man kann eine DTM angeben, die die Reduktionsfunktion $g_{TRANSFORM}()$ in polynomieller Zeit berechnet.

$FSA A_4 = DTM g_{TRANSFORM}(\langle GP \rangle) \{$

1. Prüfe, ob $\langle GP \rangle$ eine gültige Codierung eines GOTO - Programms ist. Falls nein: Gebe eine Adventure-Karte mit nicht erreichbarem Endzustand zurück.
2. Übersetze die Anweisungen des GOTO - Programms in ein Adventure der Ebene 4 wie in der nachfolgenden Tabelle beschrieben. Verbinde die einzelnen Anweisungs-Strukturen innerhalb des Adventures mit Schatz-Kanten.
3. Gebe das so erzeugte Adventure zurück.

$\}$

Anweisung	Struktur im Adventure	Erläuterungen
		Der Abenteurer muss zunächst alle Schlüssel an der Tür-Kante verbrauchen, erst wenn er keine Schlüssel mehr besitzt ($x_1 = 0$) kann er den Torbogen passieren.
		Der Abenteurer bekommt einen Schlüssel hinzu.
		Die Tür-Kante muss einmal passiert werden. Ist der Abenteurer nicht im Besitz eines Schlüssels, so kann er den Torbogen passieren (in GOTO - Programmen ist $0-1 = 0$ definiert)
		Falls der Abenteurer keinen Schlüssel hat, kann er nicht die Tür-Kante benutzen und muss daher den Torbogen durchschreiten. Umgekehrt ist er gezwungen durch die Tür zu gehen, wenn er einen Schlüssel besitzt. Um den Wert von x_1 nicht zu verfälschen, bekommt er umgehend einen neuen Schlüssel.
		Die Schatzregel (T) muss immer erfüllt sein.

		<p>Solange der Abenteurer noch Schwerter besitzt, muss er Drachen töten. Anschließend kann er durch den Fluss schwimmen.</p>
		<p>Der Abenteurer bekommt ein Schwert hinzu.</p>
		<p>Sollte der Abenteurer gerade keine Schwerter dabei haben, bleibt ihm nichts anderes übrig, als den Fluss zu durchqueren. Hat er jedoch ein Schwert, muss er den Drachen töten, da er den Fluss mit Schwert nicht passieren kann.</p>
		<p>Falls der Abenteurer keine Schwerter besitzt, muss er den Fluss durchschreiten. Ansonsten muss er sich dem Drachen stellen. Zum Glück gibt es nach jedem Drachen ein frisches Schwert, so dass der Wert von x_2 wieder auf seinen Ausgangswert gesetzt wird.</p>

Folgerung 8.2: Da das Halteproblem unentscheidbar ist, folgt dass die Frage „Ist das Adventure lösbar?“ für Abenteurer der Ebene 4 nicht entscheidbar ist.

Folgerung 8.3: Da $g_{\text{TRANSFORM}}()$ bereits angegeben wurde und GOTO - Programme eine äquivalente Definition zu den rekursiv aufzählbaren Sprachen darstellen, folgt dass es für alle $L \subseteq \Sigma^*$, $L \in RE$ ein Ebene 4 FSA $A = (Ak M, \{ (T), (G'), (D'), (R), (S) \})$ gibt, so dass gilt:

$$w \in L \Leftrightarrow g(w) \in L(A).$$

Anmerkung 8.4: Für alle FSA $A = (Ak M, \{ (T), (G'), (D'), (R), (S) \})$ gilt: $L(A) \in RE$, da es sicherlich möglich ist, eine entsprechende Turing-Maschine zu konstruieren.

Folgerung 8.5: RE entspricht $\{ \text{FSA} = (Ak, \{ (T), (G'), (D'), (R), (S) \}) \}$

9 Zusammenfassung

Zusammenfassend kann man folgendes über die Komplexität von Adventures sagen:

- Für die Ebene 0 und die Ebene 1:
 - ❖ Beide Ebenen besitzen theoretisch die gleiche Mächtigkeit, nämlich genau die der regulären Sprachen.
 - ❖ Da Adventure-Karten als NFAs gegeben sind, lässt sich die Frage „Ist das Adventure lösbar?“ in der Ebene 1 praktisch nur mit einer um $O(n^2)$ höheren Laufzeit beantworten (bzw. um $O(n)$ falls die Adventure-Regeln als DFAs anstatt als NFAs gegeben sind).
- Die Ebene 2 hat die Mächtigkeit der CFLs. Die Lösbarkeit ist mit einer Laufzeit von $O(n^4)$ entscheidbar.
- Bei der Ebene 3 kann man die Mächtigkeit mit der von Petri-Netzen vergleichen. Daher ist die Lösbarkeit des Adventures entscheidbar, die Größe der Lösung ist allerdings primitiv rekursiv
- Die Ebene 4 besitzt die Mächtigkeit von Turing-Maschinen. Es ist nicht entscheidbar, ob ein Adventure der Ebene 4 lösbar ist.

10 Literaturverzeichnis

- [1] Wilfried Brauer, Markus Holzer, Barbara König, and Stefan Schwoon. The Theory of Finite-State Adventures. *EATCS Bulletin*, 79:230-237, 2003
- [2] J. E. Hopcraft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979
- [3] Johannes Blömer. *Skript zur Vorlesung Berechenbarkeit und Formale Sprachen*. Wintersemester 2002/2003
- [4] Uwe Kastens. Vorlesungsfolien Modellierung, *Kap. 6.2. Petri-Netze* Wintersemester 2000/2001
- [5] Ekkart Kindler. *Vorversion eines Skripts zur Vorlesung Petrinetze, Version 0.0.8* Sommersemester 2002.
- [6] J. Esparza, P. Rossmanith, and S. Schwoon. A uniform framework for problems on context-free grammars. *EATCS Bulletin*, 72:169-177, October 2000.