Automatentheorie
„Berechnungsmodell für logische Sprachen“

Thorsten Haupt
Betreuer: Tim Priesnitz

-Proseminar „Theorie kommunizierender Systeme“ – SS 2004-
Prof. Gert Smolka, PS-Lab, Universität des Saarlandes

Endliche Automaten wurden unter anderem zur Modellierung von neuronalen Netzen, Schaltkreisen und Parsern eingeführt. (Kleene, 1956)
Ein Klassiker ist sicherlich die Anwendung von Automaten als Berechnungsmodell für logische Sprachen. Hier ist besonders die schwache monadische Logik 2. Ordnung mit einem Nachfolger zu erwähnen. (Büchi, 1960)
Heute werden Automaten vor allem im Compilerbau und zur Verifikation angewendet.
Nachdem in Kapitel 1 endliche Automaten formal definiert werden, sowie einige Abschluss- und Spracheigenschaften beschrieben werden, gehe ich in Kapitel 2 auf zwei Äquivalenzbegriffe sowie auf Minimierung ein. In Kapitel 3 werde ich eine Erweiterung der herkömmlichen Automaten kurz motivieren.

1. Theorie der endlichen Automaten

1.1 Einführung

Endliche Automaten werden als Tupel $(Q, \Sigma, \delta, S, F)$ definiert. Hierbei repräsentiert Q die Menge der *Zustände*, Σ das *Alphabet*, δ die *Übergangsfunktion*, S die Menge der *Startzustände* und F die Menge der *Endzustände* des Automaten.
Deterministische Automaten (DEA) und *nichtdeterministische Automaten* (NEA) unterscheiden sich in der Form der Übergangsfunktion δ. Im Falle eines DEA ist dies eine Funktion $Q \times \Sigma \to Q$. Im nichtdeterministischen Fall hat δ die Form $Q \times \Sigma \cup \{\varepsilon\} \to 2^{|Q|}$. Außerdem haben DEAs genau einen Startzustand, d.h. S ist eine einelementige Menge.
Die *Sprache* L(A) eines Automaten A ist die Menge aller Zeichenreihen die der Automat akzeptiert.
LTS (Labeled Transition Systems) unterscheiden sich von endlichen Automaten dadurch, dass es keine Start- und keine Endzustände gibt. Das Fehlen der Startzustände zeigt, dass nicht die globale Sprache, sondern die Sprache eines Zustandes von Bedeutung ist. Das Fehlen der Endzustände zeigt, dass hier die Theorie der präfixabgeschlossenen Sprachen zugrunde liegt.

In der klassischen Theorie werden Automaten als abgeschlossene Systeme betrachtet. Abbildung 1.1 zeigt wie zwei Automaten miteinander verbunden werden können. Die Kanten b und $\bar{b}$ haben einen Syn-

chronisationspunkt, der die beiden ursprünglichen Automaten verbindet.

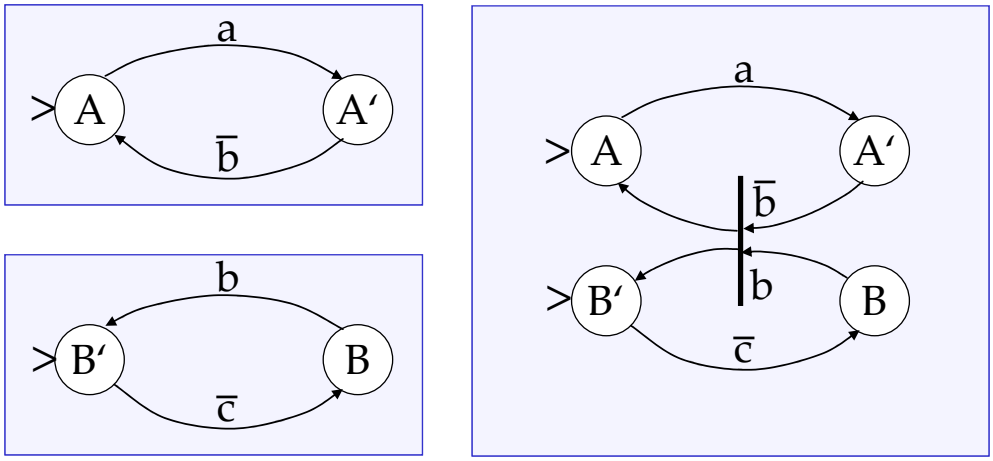


Abbildung 1.1

1.2 Determinismus

Satz (Rabin, Scott, 1959):
Jede von einem NEA akzeptierbare Sprache
ist auch durch einen DEA akzeptierbar.

Beweis siehe [3].

Man kann also grundsätzlich jeden nichtdeterministischen Automaten in einen deterministischen überführen. Die Idee hierbei ist die *Teilmengenkonstruktion*. Dazu bilden wir die Potenzmenge der Zustandsmenge des Automaten. Jedes Element dieser Menge wird zu einem Zustand im deterministischen Automaten. Es entsteht also ein Automat mit 2^n Zuständen. Die Transitionen werden danach entsprechend eingefügt. Startzustand ist der Zustand, der alle Startzustände des nichtdeterministischen Automaten enthält. Die Menge der Endzustände ist die Menge der Zustände die mindestens einen der ursprünglichen Endzustände enthält. Nicht erreichbare Zustände können entfernt werden.

Beispiel:
Für den nichtdeterministischen Automaten in Abbildung 1.2 ergibt sich aus der Teilmengenkonstruktion der deterministische Automat in Abbildung 1.3.

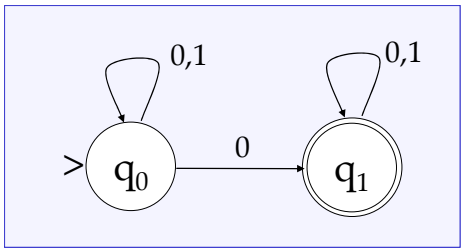


Abbildung 1.2

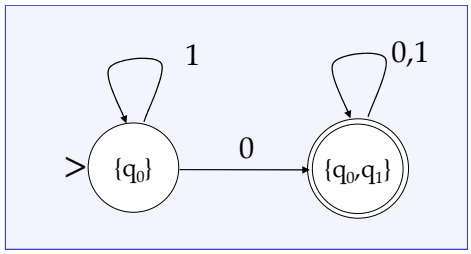


Abbildung 1.3

1.3 Abschlusseigenschaften

Nachfolgend werde ich zeigen wie man *Vereinigung*, *Komplement* und *Schnitt* von Sprachen in Automatenform repräsentieren kann. Es gibt noch einige Abschlusseigenschaften mehr (hierzu siehe [2]). Ich werde mich hier auf die oben genannten beschränken.

1.3.1 Vereinigung

Die Vereinigung zweier Automaten kann folgendermaßen gebildet werden:

- Vereinigung der Zustandsmengen
- Vereinigung der Endzustandsmengen
- Vereinigung der Transitionsmengen
- Neuer Startzustand mit ε-Übergängen zu den ursprünglichen Startzuständen

Abbildung 1.4 zeigt, wie die beiden Automaten $M_1=(Q_1, \Sigma, \delta_1, s_1, F_1)$ und $M_2=(Q_2, \Sigma, \delta_2, s_2, F_2)$ zum Automat

$$M = (Q_1 \cup Q_2 \cup \{s\}, \Sigma, \delta_1 \cup \delta_2 \cup \{ (s, \varepsilon, s_1), (s, \varepsilon, s_2) \}, s, F_1 \cup F_2)$$

vereinigt werden. M akzeptiert genau die Vereinigung der Sprachen von M_1 und M_2.

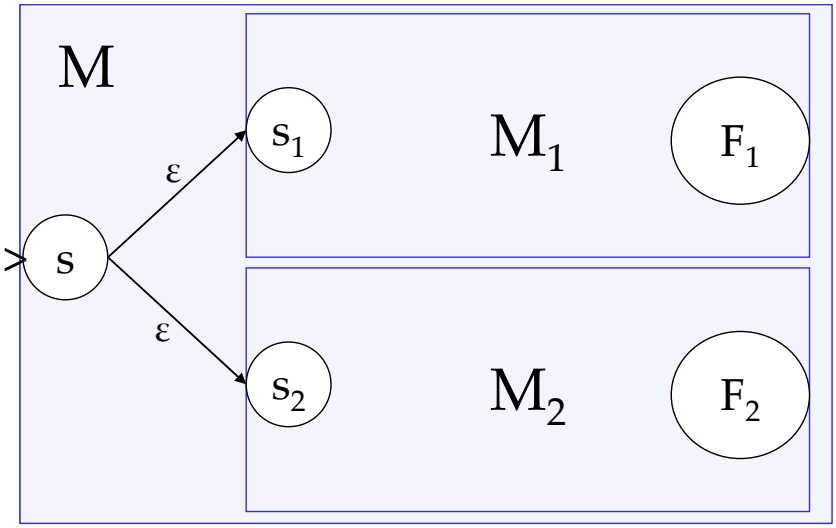


Abbildung 1.4

1.3.2 Komplement

Folgende Schritte sind erforderlich, um einen Automaten zu konstruieren, der das Komplement des Ausgangsautomaten akzeptiert:

- Automat determinisieren
- Automat vervollständigen
- Endzustände und Nicht-Endzustände tauschen

Vervollständigen heißt, dass von jedem Zustand eine Kante mit jedem Eingabesymbol ausgeht.

1.3.3 Schnitt

Um aus zwei Automaten einen Automaten zu konstruieren, der genau die Wörter akzeptiert, die beide Automaten akzeptieren bildet man den *Produktautomaten* der beiden Automaten. Hierzu sind folgende Schritte notwendig:

- Kreuzprodukt der beiden Zustandsmengen bilden.
  $\Rightarrow$ neue Zustände sind Paare
- Transitionen entsprechend übertragen.
- Endzustände sind die Zustände die nur Endzustände enthalten $\Rightarrow$ Automat akzeptiert genau dann wenn die beiden Ausgangsautomaten akzeptieren.
- Startzustand ist der Zustand, der das Paar aus den beiden ursprünglichen Startzuständen enthält.

Beispiel:
Abbildung 1.7 zeigt wie aus den beiden Automaten $M_1 = (Q_1,\Sigma,\delta_1,s_1,F_1)$ und $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ der Produktautomat

$M = (Q_1 x Q_2, \Sigma, \delta, (s_1,s_2), F_1 x F_2)$ mit $\delta((q, p), a) = (\delta_1(q, a), \delta_2(p, a))$ konstruiert wird.

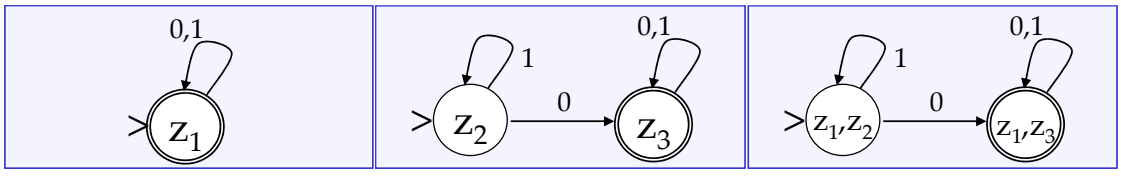


Abbildung 1.7

Die folgende Tabelle zeigt die Komplexitäten der oben gezeigten Operationen auf Automaten :

		NEA	**DEA**
Vereinigung	$L_1 \cup L_2$	$O(n)$	$O(n^2)$
Komplement	$\overline{L}$	$O(2^n)$	$O(n)$
Schnitt	$L_1 \cap L_2$	$O(n^2)$	$O(n^2)$

1.4 Spracheigenschaften

1.4.1 Leerheit

Eine Sprache ist genau dann leer, wenn es keinen Pfad von einem Startzustand zu einem Endzustand gibt. Um dies zu überprüfen genügt eine einfache Tiefensuche auf dem Zustandsgraph. Kommt man zu einem Endzustand kann man terminieren mit dem Ergebnis, dass die Sprache nicht leer ist. Dies hat die Komplexität $O(n^2)$.

1.4.2 Universalität

Eine Sprache L(A) eines Automaten A heißt universell, wenn sie alle Zeichenreihen enthält die mit Hilfe des zugrunde liegenden Σ gebildet werden können. Es muss also gelten $L(A) = \Sigma^*$.
Um auf diese Eigenschaft zu testen, können wir statt dessen testen, ob $\overline{L(A)}$ leer ist. Wir können das Problem also auf das Leerheitsproblem zurückführen.
Hierzu bilden wir das Komplement wie in 1.3.2 gesehen. Ist das Komplement leer, so ist die Sprache universell. Wie in 1.3.2 gesehen ist Determinisieren sehr teuer. Im schlechtesten Fall entsteht ein Automat mit 2^n Zuständen, wir hätten also exponentielle Laufzeit. Um dies zu vermeiden raten wir den richtigen Pfad auf dem Zustandsgraph und determinisieren „on-the-fly“. Wir determinisieren also nur den Bereich, den wir gerade betrachten. Nach $2^{|Q|}$ Übergängen können wir terminieren, da wir dann auf jeden Fall eine Transition machen, die schon einmal betrachtet wurde. Bei diesem Vorgehen erreichen wir eine Komplexitätsreduktion auf PSPACE. Das bedeutet, dass das Problem von einer Turingmaschine mit polynomiellem Speicherplatz lösbar ist.

Nachfolgend eine Übersicht über die Komplexitäten der soeben gezeigten Tests :

		NEA	**DEA**
Leerheit	$L(A) = \varnothing$?	$O(n)$	$O(n)$
Universalität	$L(A) = \Sigma^*$?	PSPACE	$O(n)$

2. Sprachäquivalenzen

In Abschnitt 2.1 werde ich erörtern, wie sich testen lässt, ob zwei Automaten in dem Sinn äquivalent sind, als sie dieselbe Sprache definieren. Eine wichtige Folge dieses Tests besteht in der Erkenntnis, dass es die Möglichkeit gibt einen DEA zu minimieren, worauf ich in Abschnitt 2.2 eingehe. Im letzten Abschnitt werde ich auf einen veränderten Äquivalenzbegriff, die Bisimulation, eingehen. Im folgenden gehe ich immer von deterministischen Automaten aus, es muss also ggf. vorher determinisiert werden.

2.1 klassische Sprachäquivalenz

Die Frage nach der Äquivalenz von Automaten basiert auf der Frage nach der Äquivalenz von Zuständen. Wann sind also zwei Zustände p und q äquivalent? Dies ist der Fall wenn folgendes gilt (vgl [2]):

> Für alle Eingabezeichenreihen w ist $\delta(p,w)$ genau dann ein akzeptierender Zustand, wenn $\delta(q,w)$ ein akzeptierender Zustand ist.

Wir fordern also nicht, dass $\delta(p,w)$ und $\delta(q,w)$ den gleichen Zustand repräsentieren, sondern nur, dass entweder beide oder keiner akzeptierende Zustände sind.
Sind zwei Zustände nicht äquivalent, so heißen sie *unterscheidbar*. Zustand p ist also von Zustand q unterscheidbar, wenn es mindestens eine Zeichenreihe w gibt, dass einer der Zustände $\delta(p,w)$ und $\delta(q,w)$ akzeptiert, der andere hingegen nicht.

Um nun die ursprüngliche Frage nach der Äquivalenz von Automaten zu beantworten müssen wir einfach überprüfen, ob die Startzustände der Automaten äquivalent sind. Ist dies der Fall, so sind auch die beiden Automaten äquivalent, repräsentieren also die gleiche Sprache.
Machen wir uns dies an einem Beispiel klar:
Abbildung 2.1 zeigt zwei Automaten mit Startzuständen A bzw. C. Wird im Startzustand 0 gelesen gehen beide Automaten in einen Endzustand über. 0 unterscheidet also die beiden Zustände nicht. Mit 1 gehen beide Automaten in einen Nichtendzustand, auch dies unterscheidet nicht. Alle Zeichen die nach einer 1 gelesen werden unterscheiden ebenfalls nicht, da entweder beide Automaten im Nichtendzustand bleiben oder beide Automaten in einen Endzustand übergehen.

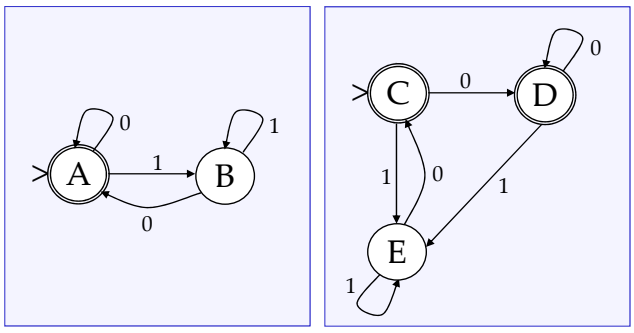


Abbildung 2.1

2.2 Minimierung

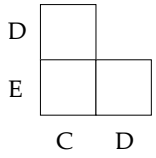
Eine wichtige Konsequenz aus der Prüfung der Äquivalenz von Zuständen ist, dass wir DEAs minimieren können. Das heißt wir können zu jedem DEA einen äquivalenten DEA finden, der so wenige Zustände hat wie kein anderer, der dieselbe Sprache akzeptiert. Dieser Minimal-DEA ist zudem für eine Sprache eindeutig (ggf. Umbenennung der Zustände). Um einen DEA zu minimieren benutzen wir den sogenannten Table-Filling-Algorithmus (vgl. [3]):

Gegeben ein Automat M = (Q, Σ, δ, s, F)

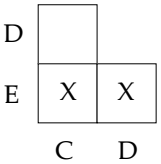
1. Stelle eine Tabelle aller Zustandspaare (*z, z'*) mit $z \neq z'$ von M auf.
2. Markiere alle Paare (*z, z'*) mit z F und $z' \notin$ F (oder umgekehrt)
3. Für jedes noch unmarkierte Paar (*z, z'*) und jedes *a* Σ teste, ob (δ(*z, a*), δ(*z', a*)) bereits markiert ist. Wenn ja, markiere auch (*z, z'*).
4. Wiederhole Schritt 3 bis sich keine Änderung in der Tabelle mehr ergibt.
5. Alle jetzt noch unmarkierten Paare können zu jeweils einem Zustand verschmolzen werden.

Schauen wir uns dieses Vorgehen am rechten Automaten in Abb. 2.1 an:

1. Wir bilden folgende Tabelle :



2. Nachdem wir die Paare mit *genau* einem Endzustand markiert haben ergibt sich folgende Tabelle:



3. Im dritten Schritt ergibt sich keine Änderung mehr, da sowohl C als auch D unter 0 in einen Endzustand gehen und unter 1 beide nach E, was also auf keinen Fall unterscheidet.

5. Hieraus ergibt sich also, dass C und D äquivalent sind und zu einem Zustand verschmolzen werden können.

Benennt man die Zustände in einem letzten Schritt noch um, erkennt man, dass der linke Automat aus Abb. 2.1 genau der Minimalautomat des rechten ist. Dies ist also ein weiterer Beweis dafür, dass die beiden Automaten äquivalent sind.

2.3 Bisimulation

Wie in [1] beschrieben, ist der klassische Äquivalenzbegriff, der alleine auf der Sprachäquivalenz beruht nicht immer passend. Dies wird an folgendem Beispiel deutlich :

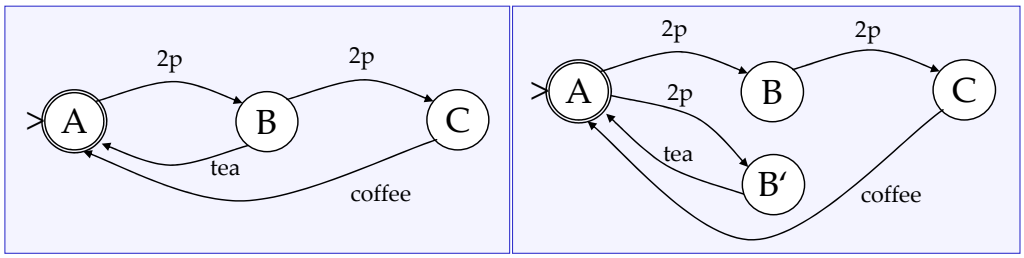


Abbildung 2.2

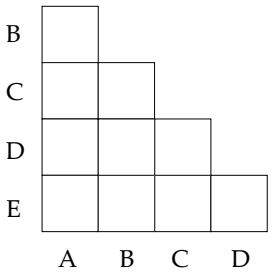
Wie sich leicht erkennen lässt akzeptieren beide Automaten die gleiche Sprache. Im ersten Fall handelt es sich um eine deterministische Repräsentation, im zweiten um eine nichtdeterministische. In der Praxis ist der zweite Automat jedoch sehr wohl vom ersten Automat zu unterscheiden.
Macht er, nachdem er 2p erhalten hat einen Übergang nach B' statt zu B, so ist der Benutzer des Automaten gezwungen Tee zu nehmen und kann nicht noch 2p einwerfen, um Kaffee zu bekommen.
Macht er einen Übergang nach B, so ist es dem Benutzer unmöglich Tee zu bekommen und er ist gezwungen weitere 2p einzuwerfen und Kaffee zu nehmen.

Wenn es darum geht wie ein Automat auf andere (hier dem Benutzer des Getränkeautomaten) reagiert, also wie er sich nach außen hin verhält, reicht Sprachäquivalenz allein nicht mehr aus.

Aus diesem Grund führen wir eine andere Form der Äquivalenz, die Bisimulation, ein. Näheres zur formalen Definition findet sich in [1], Kapitel 3. Ich möchte hier nur kurz zeigen, wie man auf Automatenebene ebenfalls durch den Table-Filling-Algorithmus auf Bisimulation testen kann.

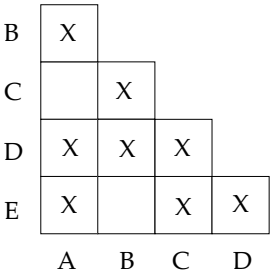
Schauen wir uns hierzu wieder Abb. 2.1 an. Wir wollen testen, ob die beiden Automaten auch unter Bisimulation als äquivalent angesehen werden können. Hierzu gehen wir folgendermaßen vor:

- Wir bilden folgende Tabelle und fassen alle Zustände der Automaten in einem *Block* {A, B, C, D, E} zusammen:

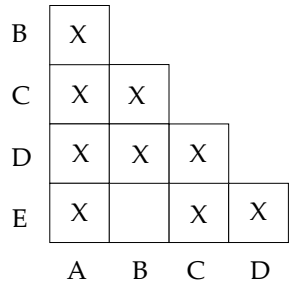


- Nun spalten wir die beiden Startzustände A und C in einem getrennten Block ab. D.h. wir haben nun zwei Blöcke {A, C} und {B, D, E}.

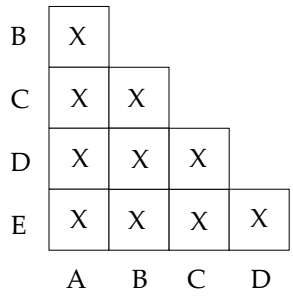
- Jetzt untersuchen wir die Transitionen zwischen diesen Blöcken. Gelangen wir unter einem Eingabesymbol *a* von manchen Zuständen eines Blocks in einen anderen Block, von anderen hingegen nicht, so unterscheidet die Transition unter *a* die Zustände des Blocks und der Block muss aufgetrennt werden.
  In unserem Beispiel betrachten wir die Kanten die mit 0 markiert sind und stellen fest, dass wir von B und E aus nach {A,C} kommen, von D aus jedoch nicht. Wir spalten also D aus {B, D, E} ab und haben nun drei Blöcke: {A,C}, {B,E} und {D}. Außerdem können wir unsere Tabelle folgendermaßen füllen:



- Betrachten wir nun den Block {D} und die Kanten mit dem Eingabesymbol 0. Von C aus kommen wir unter 0 nach D, von A aus hingegen nicht. Der Block {A, C} muss also aufgespaltet werden. Es ergeben sich also folgende Blöcke: {A}, {C}, {B,E} und {D}. Dies spiegelt sich in der Tabelle so wider:



- Im letzten Schritt betrachten wir Block {A}. Unter dem Eingabesymbol 0 kommen wir von B aus nach A, von E hingegen nicht, also müssen wir auch {B, E} aufspalten und es zeigt sich, dass keine der Zustände unter Bisimulation äquivalent sind:



Dieser Algorithmus terminiert in jedem Fall, da wir nur endlich oft teilen können. Sind zwei Zustände äquivalent, so gehören sie während des gesamten Algorithmus zu einem gemeinsamen Block. Dies ist die Invariante des Algorithmus.

Wir haben also zwei Automaten gesehen die unter der klassischen Sprachäquivalenz als äquivalent gelten, unter Bisimulation jedoch alles andere als äquivalent sind.

Folgende Tabelle zeigt abschließend die Komplexitäten der oben angegebenen Algorithmen. Minimierung und Nachweis klassischer Sprachäquivalenz sind mit anderem vorgehen auch in logarithmischer Zeit lösbar.

	NEA	**DEA**
Kl. Sprachäquivalenz	PSPACE	$O(n^2)$
Bisimulation	PSPACE	$O(n \log(n))$

3. Aussicht

Im letzten Abschnitt möchte ich nun noch kurz auf eine Möglichkeit eingehen wie man die interaktiven, kommunikativen Möglichkeiten des π-Kalküls in einer erweiterten Automatenform darstellen kann. Herkömmliche endliche Automaten können nicht mit Formalismen umgehen, deren Aktionen Informationen tragen, die in der Vergangenheit erstellt wurden. So z.B. die Kanalnamen des π-Kalküls, die von Aktionen geschaffen und später durch andere genutzt werden können.

Hierzu führen wir die so genannten HD-Automaten (History-Dependent Automata) ein. Hierbei erlauben wir, dass *Namen* explizit in Zuständen, Transitionen und Eingabesymbolen vorkommen. Diese Namen haben *lokalen* Charakter und sind damit auf den jeweiligen Zustand, die jeweilige Transition oder das jeweilige Eingabesymbol beschränkt. Jede Transition muss also die Korrespondenz zwischen den Namen der beiden Zustände und des Eingabesymbols herstellen.
Bei einem herkömmlichen Automaten sind Zustände, Transitionen und Eingabesymbole Mengen. Bei einem HD-Automaten sind es benannte Mengen. In einer Benannten Menge besitzt jedes Element eine Menge von Namen.
Bei einem Zustandsübergang eines HD-Automaten sind Ausgangszustand, Zielzustand und Eingabesymbol über drei Funktionen auf benannten Mengen miteinander verbunden. Diese mappen jeweils die Namen der an dem Zustandsübergang beteiligten Komponenten in die Transition.
Eine formale Definition von HD-Automaten findet sich in [4].
Abschließend noch ein Beispiel für eine Transition t: q $\xrightarrow{\lambda}$ q' eines HD-Automaten:

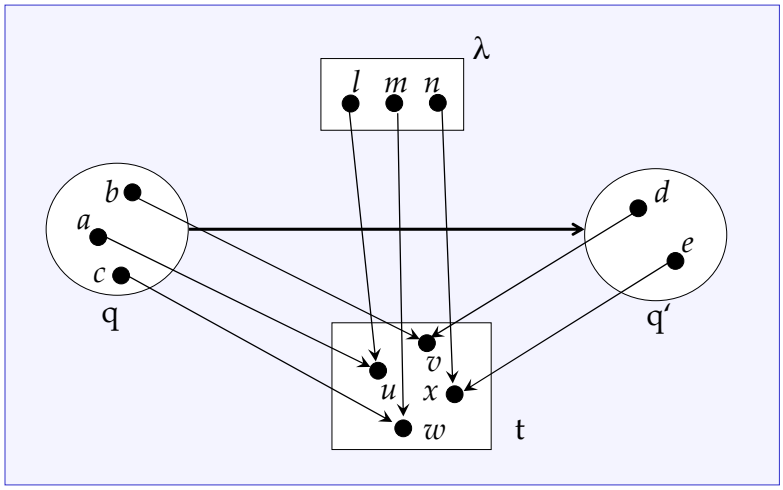


Abbildung 3.1

Referenzen

[1] Milner, R., Communicating and Mobile Systems: the π – Calculus, Cambridge University Press, 1999

[2] Hopcroft, J.E., Motwani, R. und Ullman, J.D., Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, 2. Auflage, Addison Wesley, 2002

[3] Schöning, U., Theoretische Informatik kurzgefasst, Spectrum, 2001

[4] Montanari, U. und Pistore, M., History-Dependent Automata, ENTCS, Vol. 10, 1998

[5] Hermanns, H., Verification, Vorlesung, Universität des Saarlandes, 2003