Marco Akrutat
Universität Potsdam
B. Sc. Computerlinguistik
Matr. Nr.: 771933

Endliche Automaten in der CL
Hopcroft-Projekt

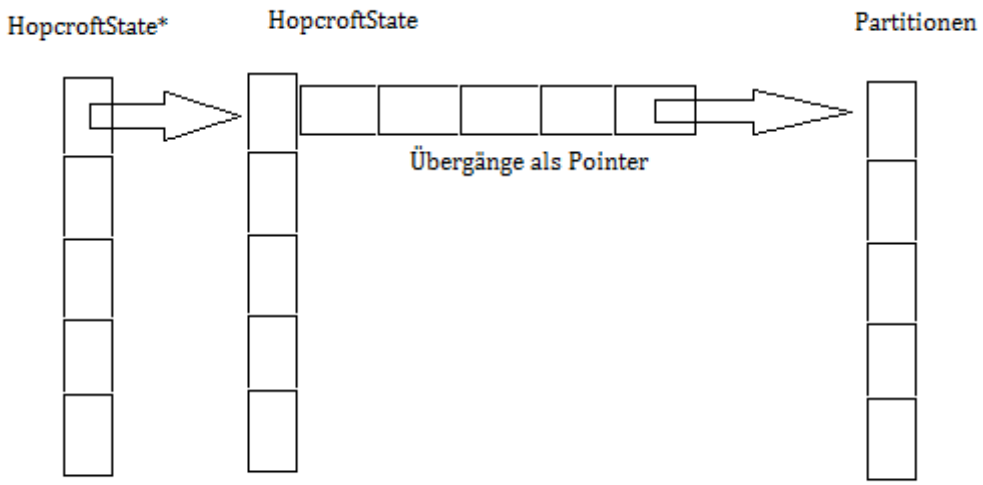
Implementierungsdetails

In diesem Projekt minimiere ich deterministische endliche Automaten (DEA). Ich werde bei der Erklärung des Algorithmus nicht ins Detail gehen, sondern mich auf die Besonderheiten meiner Implementierung konzentrieren. Für jeden DEA gibt es einen einzigartigen minimalen DEA, der die gleiche Sprache akzeptiert und dabei eine minimale Anzahl an Zuständen besitzt. Um diesen minimalen DEA zu finden benutze ich einen Algorithmus von Hopcroft(1971). Dieser Algorithmus partitioniert die Zustände in Gruppen von Zuständen die die gleichen Rechtssprachen akzeptieren, und demnach Equivalent sind.

Die Partitionierung der Zustände beginnt mit zwei Gruppen. Den Endzuständen und Nicht-Endzuständen. Nun werden die Zustände einer Gruppe betrachtet, und festgestellt in welcher Gruppe die Zustände sind, zu denen sie einen Übergang mit einem bestimmten Alphabetssymbol haben. Dabei werden die Zustände so in neue Gruppen verteilt, dass sie für alle Symbole das gleiche Verhalten aufweisen. Dieser Prozess wird fortgeführt bis nach einem Schleifendurchlauf keine Veränderung der Gruppen mehr auftritt. Die Zustände innerhalb einer Gruppe können nun zusammengelegt und so der neue minimale DEA konstruiert werden.

Um eine Schleife über die Alphabetssymbole zu vermeiden, verfügt jeder Zustand in meinem DEA über eine Signatur, ein *long*, das Informationen enthält für welche Alphabetssymbole ein Zustand Übergänge enthält. Das *long* funghiert hier als Boolvektor mit 64 Bits. So beschränkt sich das Alphabet in meiner Implementierung auf max. 63 Zeichen, da ich das letzte Bit zur Endzustandsmarkierung verwende. Die Minimierung wird in 4 Schritten ausgeführt: *init(), first_step(), refine()* und *construct()*. Im *init()* Schritt werden die nötigen Datenstrukturen initialisiert.

Bild 1:



Den Kern bildet hier der Vektor aus *HopcroftStates* in der Mitte in Bild 1. Diese enthalten die Zustandsnummer sowie einen Vektor aus Pointern, die auf die Partitionen zeigen, in denen sich die Zielzustände befinden. Diese befinden sich in einem Vektor, in dem jeder Index einem Zustand entspricht und das gespeicherte *unsigned int* seiner Partition. Mithilfe dieser Architektur, kann ich die Partition eines Zustandes verändern, und alle Übergänge, die zu diesem Zustand führen wissen automatisch Bescheid. Während der Initialisierung der Datenstrukturen werden bereits alle Endzustände und alle Nicht-Endzustände in jeweils eine Partition getan.

Nun basieren alle folgenden Abläufe darauf die *HopcroftStates* zu sortieren, und anschließend über den Vektor zu iterieren und die Partitionen zu verfeinern. Deshalb gibt es den Vektor mit Pointern zu den *HopcroftStates* ganz links in Bild 1. Um beim Sortieren nicht tatsächlich die *HopcroftStates* und den in ihnen enthaltenen Vektor bewegen zu müssen, wird der Vektor mit den Pointern sortiert und über diesen iteriert.

Im zweiten Schritt *first_step()* wird nach folgenden Kriterien sortiert: Partitionen > Signatur > Übergänge. Die Signaturen sorgen dafür, dass alle Zustände mit gleichen Übergängen hintereinander sind, und diese wiederum sind nach dem Verhalten ihrer Übergänge geordnet. So können bei der folgenden Iteration über den Vektor alle Zustände aus derselben Partition, mit der gleichen Signatur, und sich gleich verhaltenden Übergängen zusammen gruppiert werden. Dank dem Sortieren befinden sich die Zustände jeder neuen Gruppe im Vektor hintereinander.

Der *refine()* Schritt funktioniert auf dieselbe Art und Weise, allerdings brauch die Signatur beim Sortieren und Iterieren nichtmehr beachtet zu werden, da nun mit Sicherheit nurnoch Zustände mit gleichsymboligen Übergängen innerhalb einer Gruppe sind. Dies geschieht in einer while-Schleife solang bis keine Änderung mehr erfolgt.

Im *construct()* Schritt wird nun der minimale DEA konstruiert und zugewiesen. Der Vektor der Partitionen fungiert hier als *map,* die die alten Zustände auf ihre Neuen abbildet.

Literatur:

Hopcroft, John (1971), "An *n* log *n* algorithm for minimizing states in a finite automaton", *Theory of machines and computations (Proc. Internat. Sympos., Technion, Haifa, 1971)*, New York: Academic Press, pp. 189–196, MR 0403320