Dokumentation zum Konvertierer und Interpreter für Turingmaschinen

Sven Fiergolla

November 26, 2017

Einführung

Wie in "A Universal Turing Machine with Two Internal States" von Claude E. Shannon beschrieben, lässt sich jede Turingmaschine in eine TM mit nur zwei Zuständen überführen.
Im folgenden wird die Impementierung, der im Paper beschrieben Konvertierung, dokumentiert und erläutert. Zudem wird der Interpreter erläutert.

Funktionalität & Abhängigkeiten der verwendeten Klassen

Konvertierer (Package `construction`)

Für die Konvertierung einer TM nach dem beschrieben Verfahren werden eine Reihe neuer Symbole benötigt, da die Information des aktuellen Zustandes in die Symbole übertragen wird. Anschließend müssen die Übergänge der TM und das Startsymbol (das Symbol unter dem Lesekopf zum ersten Schritt) angpasst werden. Dazu existieren die Klassen `ComplexSymbol` und `TMConstructor`.

`ComplexSymbol`

Nach Shannons Verfahren, müssen zu jedem elementaren Symbol der ursprünglichen TM, pro Zustand, 4 neue Symbole erstellt werden. Dazu werden die Konstanten aus der Dummy-Klasse `ComlexSymbol` verwendet.

`TM2Generator`

Die Klasse `TM2Generator` kann mit Hilfe der Funktion `readTMfromFile(String path)` bzw. über den Konstruktor, eine Turingmaschine im beschrieben `.tur`-Format einlesen. Nun lässt sich die Funktion `generate2StateTM()` anwenden, welche 4 weitere Funktionen aufruft:

`generateComSymbolTable()`

Die von der neuen TM benötigten Symbole werden pro Symbol von der Methode `generateSymbolArray (String symbol, String[] states)` mit Hilfe von `ComplexSymbol` für alle Zustände in je 4 Variationen erstellt und anschließend im `String[][] compSymbolTable` gehalten. Sind $A_1, A_2 \ldots A_m$ die Symbole der ursprünglichen TM und $q_0, q_1 \ldots q_n$, so ist das Array `compSymbolTable` anschließen wie folgt aufgebaut:

$A_{0,q_0,-,R}$	$A_{1,q_0,-,R}$	$\ldots$	$A_{m,q_0,-,R}$
$A_{0,q_0,-,L}$	$A_{1,q_0,-,L}$	$\ldots$	$A_{m,q_0,-,L}$
$A_{0,q_0,+,R}$	$A_{1,q_0,+,R}$	$\ldots$	$A_{m,q_0,+,R}$
$A_{0,q_0,+,L}$	$A_{1,q_0,+,L}$	$\ldots$	$A_{m,q_0,+,L}$
$A_{0,q_1,-,R}$	$A_{1,q_1,-,R}$	$\ldots$	$A_{m,q_1,-,R}$
$A_{0,q_1,-,L}$	$A_{1,q_1,-,L}$	$\ldots$	$A_{m,q_1,-,L}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$A_{0,q_n,+,L}$	$A_{1,q_n,+,L}$	$\ldots$	$A_{m,q_n,+,L}$

`generateNativeTransitions()`

Damit die neue TM im wesentlichen wie die ursprüngliche TM agiert, werden die alten Übergänge modifiziert. Nach Shannon's Konstruktion muss folgende Gleichung gelten:

$$\delta(A_i, q_j) \rightarrow (A_k, q_l, \genfrac{}{}{0pt}{}{R}{L}) \Rightarrow \delta(B_{i,j,-,x}, \alpha) \rightarrow (B_{k,l,+,\genfrac{}{}{0pt}{}{R}{L}}, \genfrac{}{}{0pt}{}{\beta}{\alpha}, \genfrac{}{}{0pt}{}{R}{L}) \tag{1}$$

Jeder ursprüngliche Übergang aus dem `transitions`-Array wird wie folgt bearbeitet:
Zu Beginn werden die Indizes der Symbole und Zustände analysiert, sodass bekannt ist, welche Position die Symbole und Zustände in `sigma` und `states` haben. Ist der alte Übergang aus Zustand q_j mit Symbol A_i, so ist der neue Übergang aus dem Symbol im `compSymbolTable` der Zeile $4 \cdot j$ und $4 \cdot j + 1$ (da der Übergang aus beiden Symbolen mit $-$ im Index exisitiert) und Spalte i.
Je nach Richtung des alten Übergangs wird nun ein neuer Übergang erstellt. Ist die Richtung `R`, so wird das neue Symbol `R` im Index haben, die Richtung des Übergangs ist ebenfalls `R` und der Folgezustand wird β sein (analog bei ursprünglicher Kopfbewegung nach links).
Das auszugebende Symbol wird ähnlich aus der `compSymbolTable` ermittelt. Ist das auszugebden Symbol im alten Übergang A_k und der Folgezustand q_l , so wird für den neuen Übergang das passende auszugebende Symbol aus der Tabelle der komplexen Symbole ermittelt. Es befindet sich in der Zeile $4 \cdot l + 2$ wenn der ursprüngliche Übergang mit Kopfbewegung nach rechts ist (sonst in der Zeile $4 \cdot l + 3$) und in der Spalte k.
Anschließend sind die Übergänge äquivalent von der TM mit nur 2 Zuständen zu realisieren.

`generateCompTransitions()`

Die neue TM benötigt eine Reihe Hilfsübergänge, um elementare Symbole in komplexe umzuwandel und umgekehrt sowie für die sogenannte "bouncing operation", der von Shannon beschriebenen Vorgehensweise die Information des Zustandes in Symbole auslagern und diese Information, zwischen den Feldern des Bandes einer TM, zu verschieben.
Die benötigten Übergänge lauten:

Gleichung	**Symbol**	**Zustand** $\Rightarrow$	**Symbol**	**Zustand**	**Richtung**
(1)	B_i	α	$B_{i,1,-,R}$	α	R
(2)	B_i	β	$B_{i,1,-,L}$	α	L
(3)	$B_{i,j,-,x}$	α oder β	$B_{i,(j+1),-,x}$	α	$x \in \{R, L\}$
(4)	$B_{i,j,+,x}$	α oder β	$B_{i,(j-1),+,x}$	β	$x \in \{R, L\}$
(5)	$B_{i,1,+,x}$	α oder β	B_i	α	$x \in \{R, L\}$

Nach Gleichung (1) exisitiert ein Übergang zwischen jedem elementaren Symbol B i in Zustand α, zu dem komplexem Symbol mit gleichem Index welches sich in der ersten Zeile der Tabelle befindet, mit Folgezustand α und Richtung R.
Nach Gleichung (2) exisitiert ein Übergang zwischen jedem elementaren Symbol B_i in Zustand β, zu dem komplexem Symbol mit gleichem Index welches sich in der zweiten Zeile der Tabelle befindet, mit Folgezustand α und Richtung L.
Nach Gleichung (3) exisitiert ein Übergang zwischen einem komplexen Symbol mit $-$ im Index aus Zustand β, zu dem komplexen Symbol in der gleichen Spalte und 4 Zeilen weiter mit Folgezustand α. Dies gilt für beide Richtungsinzes `R` und `L`.
Nach Gleichung (4) exisitiert ein Übergang zwischen einem komplexen Symbol mit $+$ im Index aus Zustand α oder β, zu dem komplexen Symbol in der gleichen Spalte und 4 Zeilen zurück mit Folgezustand β. Dies gilt für beide Richtungsinzes `R` und `L` und natürlich nur für Symbole ab Zeile 4.
Nach Gleichung (5) exisitiert ein Übergang zwischen einem komplexem Symbol aus der 1. und 2. Spalte, aus Zustand α oder β, zu dem elemntaren Symbol mit gleichem Index. Abhängig vom Zustand ist die Richtungsänderung `R` oder `L`.
Alle modifizierten nativen und Hilfsübergänge werden in der Liste `transitionsNew` gespeichert und nach erfolgreicher Konvertierung in die neue `.tur`-Datei geschrieben.

`modifyInitialSymbol()`

Interpreter

`State`

`Tape`

`Transition`

`TuringMachine`

Application

Dateiformat `.tur`

Der Versuch einen Konverter für Turingmaschinen, für einen bereits existierenden Simulator für TM's, zu entwerfen erwies sich als schwierig, da die verwendeten Konventionen und Notationen stark von den in der Vorlesung verwendeten abweichen. So wurde das Format `.tur` für das einfache Einlesen und Konvertieren von TM's konzipiert.

Beispieldatei `equal01.tur`

```
states
q0
q1
q2
q3d
q4
q5a

transitions
q0 q0 0 0 L
q0 q0 1 1 L
q0 q1 # # R
q0 q0 X X L
q1 q2 0 X R
q1 q4 1 X R
q1 q1 X X R
q1 q5a # # L
q2 q0 0 0 R
q2 q0 1 X L
q2 q2 X X R
q2 q3d # # R
q4 q0 0 X L
q4 q4 1 1 R
q4 q4 X X R
q4 q4 # # R

symbols
0 1 X #

tape
[ 0 ] 1 1 1 0 0

description
This TM evaluates if the given input contains an equal ammount
of zeros and ones.
```

Anwendung