Dokumentation zum Konvertierer und Interpreter für Turingmaschinen

Sven Fiergolla

December 6, 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:

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

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 ausulagern 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()`

Shannon nennt in seinem Paper bei der Beschreibung des Beispiels als induktive Annahme, dass das Symbol unter dem Lesekopf bereits ein komplexes ist. Dies bedeutet, dass das erste Symbol der neuen

TM mit nur 2 Zuständen, im Startsymbol bereits die Information über den Aktuellen Zustand hält. Ist dies nicht der Fall, so läuft die TM endlos nach links auf der Suche nach dem ersten komplexen Zeichen welches Information über den aktuellen Zustand innehält.
Folglich muss das Symbol unter dem Lesekopf beim Start der TM angepasst werden, in das äquivalente komplexe Symbol, welches den Startzustand der ursrünglichen TM behält. Dies ist nach Konvention des `.tur`-Formats das erste Symbol der `compSymbolTable`.

Interpreter

Um die Turingmaschinen vor und nach der Modifizierung ausführen beziehungsweise visualisieren zu können, ist ein Interpreter für Turingmaschinen von nöten, welcher das Dateiformat interpretieren kann. Dazu exisitieren die folgenden Klassen.

`State`

Ein State kapselt im wesentlichen nicht mehr als den Namen des Zustands und die Information, ob der Zustand final, wenn ja, akzeptierend oder ablehnend ist. Nach Konvention endet der Name eines Finalzustandes mit f, der Name eines ablehnenden Zustandes mit d (für *decline*) und der Name eines akzeptierenden Zustandes mit a (für *accept*).
Diese Konvention wird auch bei der TM mit nur 2 Zuständen genutzt. In dem Fall sind die beiden Zustände der Maschine α und β nicht final, jedoch ist die Information des aktuellen Zustandes in die Symbole kodiert. Hat eine solche TM ein Eingabeband abgearbeitet, erreicht sie ein Symbol für das sie keinen Übergang besitzt (falls in der original TM aus dem final/akzeptierenden/ablehnenden Zustand kein Übergang exisitiert). In diesem Fall lässt sich der in das Symbol eingebettete Zustand analysieren und es lässt sich entscheiden, ob und welcher Finalzustand äquivalent ist. So lässt sich auch bei der Simmulation der TM mit nur 2 Zutsänden das Terminieren der TM erkennen und auswerten.

`Tape`

Das Band der simmulierten TM wird durch die Klasse `Tape` realisiert. Sie speichert das aktuelle Symbol sowie die linke und rechte Seite des Bands durch zwei Stacks. Zudem werden die Basisoperationen Kopfbewegung nach `L` oder `R` unterstützt.
Das Lesen und Schreiben eines Tapes aus einer Datei wird ebenfalls durch diese Klasse realisiert.

`Transition`

Eine Instanz der Klasse `Transition` speichert den Folgeschritt aus einem Übergang, sprich den neuen Zustand, das auszugebende Symbol und die Richtung der Kopfbewegung dieses Übergangs. Damit ist diese Kalsse ebenfalls reine Kapselung von Daten und hat an sich keinerlei eigene Funktionalität.

`TuringMachine`

Die eigentliche TM und deren simmulation wird durch die Klasse `TuringMachine` realisiert. Sie besteht aus einer Reihe statischer Indizes, einem aktuellen Zustand gespeichert in `currentState`, einem Tape und einer Übergangstabelle. Diese wird durch eine doppelt verschachtelte Map realisiert, sodass einem Zustand, eine Reihe eingabe Symbole zugeordnet sind, und jeder Zustands/Eingabesymbol-Kombination nur ein Übergang in Form einer `Transition`. Die Listen `history` und `historyDetails` speichern zudem den Verlauf und genaue Einzelheiten des Ablaufs der TM.
Neben einfachen Utilities-Methoden die das Lesen und Schreiben einer TM aus einer Datei ermöglichen, hat die Klasse die folgenden wichtigen Funktionen:

`setTransitionMap(String[] states, String[] transitionsArray)`

Diese Methode erstellt aus der eingelesenen TM die bereits beschriebene, doppelt verschachtelte (Hash)Map aus Zustand, eingelesenes Symbol $\rightarrow$ Übergang. Dazu werden alle Übergänge nacheinander in die Map eingefügt. Wird ein neuer Übergang an die Position eines bereits exisitierenden Übergangs eingefügt, ist die zu simmulierende TM nicht deterministisch und der Prozess wird abgebrochen.

`step()`

Die Methode `step()` reslisiert einen Schritt der TM. Mit dem aktuellen Zustand und Symbol wird aus der `transitions` Map der passende Übergang herausgesucht. Anschließend wird das aktuelle Symbol verändert und eine Kopfbewegung und Zustandsänderung ausgeführt.

`run(long timeoutMili)`

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.
```

Das Dateiformat beseteht aus einer Auflistung aller in der TM vorkommenden Zustände, beginnend mit dem Startzustand, eingeleitet durch das flag `states`, getrennt durch Zeilenumbruch. Darauf folgt eine Liste der Übergänge, eingeleited durch `transitions` und eine Zeile aller Symbole welcher das flag `symbols` voraus geht. Für die Simulation kann ebenfalls ein Band beschrieben werden, es besteht aus einer Aneinanderreihung von Symbolen, getrennt durch whitespace. Das Symbol, welches zu Beginn der Simualtion unter dem Lesekopf sein soll, ist von eckigen Klammern umgeben, ebenfalls getrennt durch whitespace.
Weitere Informationen über das Dateiformat lassen sich der Datei `FORMAT_EXAMPLE.tur` entnehmen.

Anwendung

Die fertige Anwendung lässt sich als `jar` wie folgt benutzen:
`<.tur Datei> <delay in millisekunden>` (optional, ohne Verzögerung ist es jedoch schwer der Simulation zu folgen)
Weitere Parameter die gesetzt werden können:
`-c` konvertiert die TM in eine TM mit 2 Zuständen vor der Simulation
`-p` speichert Simulation in einer .history Datei ab
`-pd` speicher zudem Details wie aktueller Zustand und Übergang
`-d` debug (Genaurere Informationen des Ablaufs werden über st.out ausgegeben)