Dokumentation zum Konvertierer und Interpreter für Turingmaschinen

Sven Fiergolla

November 23, 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 $A_1, A_2 \dots A_m$ die Symbole der ursprünglichen TM und $q_0, q_1 \dots q_n$, so ist das Array `compSymbolTable` anschließen wie folgt aufgebaut:

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

`generateCompTransitions()`

`generateNativeTransitions()`

`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