Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Das Video zur Lösung der Aufgabe 8a finden Sie unter: https://youtu.be/1qx6II810Ng

Das Video zur Lösung der Aufgabe 8b finden Sie unter: https://youtu.be/umY2HsIf48I .

```
für n=0 ist der Start nicht ok, korrigiert (erste Zeile streichen):
Start # suche0 # L
suche0 c sucheb # L
suche0 # S # S
weiterhin fehlen:
sucheb B sucheb B L
neuesc B neuesc B R
neuesB A neuesB A R
```

Frage: Bei der nachträglichen Bearbeitung der optionalen Zusatzaufgaben ist mir bei Aufgabe 2d eine Frage aufgekommen: Warum würde der Operator "==" theoretisch anstelle von ".equals()" funktionieren? Muss im Umkehrschluss jedes Zeichen immer dieselbe Speicheradresse haben, und wenn ja, an welcher Stelle in der Zeichen.java wurde dies so festgelegt?

Antwort: Es ist wie sie es beschreiben. Es ist eine Design-Entscheidung, dass jedes Zeichen-Objekt eindeutig sein soll und dies wird mit einem Singleton (s. auch OOAD) erreicht.
Für Details ist es wichtig, dass z. B. Leerzeichen über /space als Zeichen verarbeitet werden sollen. Deshalb gibt es eine Map<Name des Zeichens, Zeichen>, wobei rechts das eindeutige Zeichen (Singleton) steht. In fast allen Fällen wird einem normalen Buchstaben wie "x" genau das Zeihen x zugeordnet. Der Konstruktor von Zeichen ist nicht nach außen sichtbar, es gibt dafür mit public static Zeichen zeichen(String z) eine Build-Methode, mit der sichergestellt wird, dass jedes Zeichen eindeutig ist.

Frage: Beim Quiz 3 zur Lernnotiz 2 ist

☐ $f1(I^n) = f1(I^{n+1})I$ (das letzte I bedeutet, es wird angehängt) nicht berechenbar, warum?

Antwort: Dies ist keine ordentliche Funktionsdefinition und deshalb nicht berechenbar.

$f1(\varepsilon) = f1(I)I$, d.h. das Ergebnis von $f1(\varepsilon)$ hat mindestens ein I am Ende,

da $f1(I) = f1(II)I$, d. h. $f1(I)$ hat mindestens ein I am Ende, d. h. aber $f1(\varepsilon)$ hat mindestens II am Ende.

Wird dies schrittweise fortgeführt, muss $f1(\varepsilon)$ dann unendlich viele I am Ende haben, was als Ergebnis einer Turing-Maschine nicht zugelassen ist.

Frage: Wie funktioniert die universelle Turing-Maschine?

Antwort: Wichtig ist die Idee dahinter. Sie wollen eine Turing-Maschine schreiben die Turing-Maschinen ausführt, ähnlich zu einem Simulator für Turing-Maschinen, nur dass dieser selbst als

Turing-Maschine geschrieben wird. Der Ansatz ist genau wie für einen Simulator, als Eingabe wird die Turing-Maschine, die ausgeführt werden soll und das Eingabewort auf das Band geschrieben. Dann startet die universelle Turing-Maschine, schreibt zunächst die Ausgangskonfiguration der auszuführende Turing-Maschine auf das Band, liest das Zeichen unter dem Schreiblesekopf und den Zustand und sucht dann auf dem Band in der auszuführenden Turing-Maschine nach dem Folgezustand und dem neuen Bandinhalt. Jede dieser Aktionen benötigt einige Zustände und Hilfszeichen, um sich Details zu merken. Wichtig ist aber, dass nur endlich viele Zustände und endlich viele Zusatzzeichen benötigt werden, da für jede Turingmaschine die Kombination aus Zeichen und Zuständen immer endlich ist. So kann letztendlich garantiert werden, dass es eine universelle Turing-Maschine gibt.

Frage: Wenn ich es richtig verstehe, sind die Begriffe „rekursiv aufzählbar“ und „aufzählbar“ Synonyme?

Antwort: Ja, es werden manchmal in den Folien mehrere Begriffe angegeben, die das Gleiche bedeuten, da sie diese auch in anderen Lehrbüchern in diesen Varianten finden. Der Begriff „semi-entscheidbar“ ist hier ein weiteres Synonym.

Frage: Was ist die Halteproblemsprache genau?

Die Halteproblemsprache besteht aus allen Wörtern, die die folgende Form haben: Turingmaschine#Eingabewort und die Turingmaschine hält angesetzt auf das Eingabewort an. Wir wissen dazu, dass eine Turingmaschine auch nur aus endlich vielen Zeichen besteht und so auf das Band geschrieben werden kann.

Bei der speziellen Halteproblemsprache wird davon nur die Teilmenge betrachtet, bei der das Eingabewort wieder die Turing-Maschine selbst ist, also die Eingabe der Form TMX#TMX hat.

Eine weitere oft betrachtetet Variante beschäftigt sich mit der Sprache, die als Worte alle Turing-Maschinen beinhaltet, die angesetzt auf das leere Wort anhalten.

Für alle diese Sprachen gilt, dass sie nicht entscheidbar, aber aufzählbar sind.

Frage: Wie hängen jetzt charakteristische Funktion und Entscheidbarkeit zusammen?

Antwort: Erstmal sind charakteristische Funktionen unabhängig von Turing-Maschinen und Entscheidbarkeit definiert. Für eine beliebige Menge M gilt f(x) = 1, wenn x in M ist, sonst f(x) = 0.

Die charakteristische Funktion kann aber auch mit Hilfe einer Turing-Maschine berechnet/programmiert werden. Da gilt dann, dass genau dann, wenn M entscheidbar ist, ist die charakteristische Funktion Turing-berechenbar.

Frage: Was ist der Busy Beaver genau?

Antwort: Die Aufgabe ist: Sie können n Zustände nutzen und müssen eine Turing-Maschine schreiben, die möglichst viele Striche auf dem Band als Ergebnis liefert. Das bedeutet natürlich auch, dass die Turing-Maschine anhalten muss. Die Frage ist dann nach der zu n gehörenden maximalen

Strichanzahl, wobei es zwischen den Strichen Leerzeichen geben darf. Diese Funktion ist nicht berechenbar.

Frage: Ich habe mit der Diagonalisierung bei dem Halte-Problem ein Problem.

Antwort: Schauen Sie sich nochmals die Idee der Überabzählbarkeit der reellen Zahlen an, die Idee ist die gleiche. Grob ist der Ansatz immer, wenn es ein Verfahren geben soll, dann gibt es auch ein zweites Verfahren, das recht einfach auf dem ersten basiert und damit auch existiert. Dann wird gezeigt, dass das zweite Verfahren für einen bestimmten Wert zwei verschiedene Ergebnisse liefern müsste, was nicht geht. Damit kann das zweite und damit auch das erste Verfahren nicht existieren.

Frage: Welches Wort ist $a^nb^{2n}c^{3n}$ für n=0?

Antwort: „hoch 0“ ergibt für jedes Zeichen und Wort immer ε, so dass das auch das Gesamtergebnis ist.

Frage: Welches Wort ist $a^nb^{2n}c^{3n}$ für n=-1?

Antwort: Das ist nicht definiert, da unsere Definition mit n=0 startet. [Man kann auch darüber nachdenken und interessante Konstruktionen basteln, aber das ist eher reine Mathematik jenseits der Theoretischen Informatik. Wer sich dafür genauer interessiert, unsere Sprachen mit der Konkatenationsverknüpfung bilden ein freies Monoid. Rest ist Algebra.

Frage: Wie gehören die Begriffe Entscheidbarkeit, Aufzählbarkeit und Akzeptierbarkeit zusammen?

Akzeptierbarkeit einer Sprache L ist zunächst die Forderung, dass es eine Turing-Maschine gibt, die irgendwann anhält, wenn sie auf ein Wort der Sprache L angesetzt wird.

Dies ist eng verwandt mit dem Begriff der Aufzählbarkeit, die nur die zusätzliche Form der Ergebnisausgabe der Turing-Maschine fordert. Wenn es für L eine akzeptierende Turing-Maschine gibt, ist es trivial daraus eine aufzählende Turing-Maschine konstruieren (erste Diagonalisierungsidee, alle Worte schrittweise ausprobieren). Genauso kann aus der aufzählenden Turing-Maschine eine akzeptierende gebaut werden, es werden mit der gleichen Diagonalisierungsidee alle Turing-Maschinen mit I^n als Eingabe ausgeführt und angehalten, wenn das Wort gefunden wurde. Etwas unsauber kann damit bei Aufzählbarkeit und Akzeptierbarkeit von Synonymen gesprochen werden.

Die Entscheidbarkeit ist anders, da für eine Sprache L die Turing-Maschine irgendwann mit einem bestimmten Ergebnis anhalten muss, wenn das Wort in L liegt und mit einem anderen Ergebnis anhalten muss, wenn das Wort nicht in L liegt. Das „und“ macht es anders, da so die Akzeptierbarkeit bzw. Aufzählbarkeit von L (wie vorher auch) und zusätzlich die Akzeptierbarkeit bzw. Aufzählbarkeit des Komplements von L gefordert wird.

Damit ist L entscheidbar, wenn L akzeptierbar und auch das Komplement von L akzeptierbar ist.

Frage: Ich finde das Thema mit der Entscheidbarkeit ziemlich schwer, wie sehen Sie das bei den anderen Themen.

Antwort: Vom Inhalt her haben Sie Recht, das ist das schwerste Kapitel der Veranstaltung. Die nachfolgenden sind meist deutlich leichter. Ich habe das Thema an den Anfang der Veranstaltung gestellt, da die Aussage „es gibt viele Eigenschaften von Programmen, die wir nicht automatisch nachweisen können“ die zentrale Aussage der Veranstaltung ist, die gerade auch in der Praxis im Hinterkopf bleiben muss. (Weiterhin sind Sie ja am Anfang des Semesters noch am Frischesten :))

Frage: Gibt es auch zu den anderen Aufgaben Beispiellösungen?

Antwort: Ist aktuell nicht geplant. Sie können aber Ihre Lösungen mit mir besprechen (VL-Zeit und Übungszeit). Das ist auch möglich, wenn Sie keinen Ansatz gefunden haben.

Frage: Gibt es für die Reduktionen ein Kochrezept?

Antwort: Ganz allgemein leider nicht. Trotzdem besteht immer die Idee mit einer Turing-Maschine ein Problem der einen Art in ein Problem der anderen Art zu verwandeln. Statt der formalen Forderung:

Definition: Gegeben seien Sprache1 und Sprache2 nicht notwendigerweise über den gleichen Alphabeten Alphabet1 und Alphabet2 sowie eine total berechenbare Funktion
f: Alphabet1* -> Alphabet2*; gelte dann für alle $w \in$ Alphabet1*
$w \in$ Sprache1 genau dann wenn $f(w) \in$ Sprache2,

Kann man auch äquivalent etwas anschaulicher versuchen zu zeigen:

$w \in$ Sprache1 daraus folgt $f(w) \in$ Sprache2 und

$w \notin$ Sprache1 daraus folgt $f(w) \notin$ Sprache2

das ist logisch äquivalent zu:

$w \in$ Sprache1 daraus folgt $f(w) \in$ Sprache2 und

$f(w) \in$ Sprache2 daraus folgt $w \in$ Sprache1

Minibeispiel: Sprache1={aa, bb} und Sprache2={xx}, dann wird f so konstruiert, dass bei der Eingabe von aa oder bb die Eingabe gelöscht und xx auf das Band geschrieben wird; bei anderen Eingaben wird yy ausgegeben, dann

a) $w \in$ Sprache1 daraus folgt $f(w) \in$ Sprache2, gilt da $aa \in$ Sprache1 daraus folgt $f(w)=xx \in$ Sprache2 und $bb \in$ Sprache1 daraus folgt $f(bb)=xx \in$ Sprache2 und für alle anderen w ist f(w) nicht in Sprache2
b) $f(w) \in$ Sprache2 daraus folgt $w \in$ Sprache1, gilt da $f(aa) = xx \in$ Sprache2 daraus folgt $aa \in$ Sprache1 und $f(bb) = xx \in$ Sprache2 daraus folgt $bb \in$ Sprache1, weiterhin wird kein anderes Wort aus der Sprache2 durch f erreicht

Variante: Sprache1={xx} und Sprache2={aa,bb}, dann wird f so konstruiert, dass bei der Eingabe von xx die Eingabe gelöscht und aa auf das Band geschrieben wird; bei anderen Eingaben wird yy ausgegeben, dann

a) w $\in$ Sprache1 daraus folgt f(w) $\in$ Sprache2, gilt da xx $\in$ Sprache1 daraus folgt f(w)=aa $\in$ Sprache2 und für alle anderen w ist f undefiniert
b) f(w) $\in$ Sprache2 daraus folgt w $\in$ Sprache1, gilt da f(xx) = aa $\in$ Sprache2 daraus folgt xx $\in$ Sprache1, weiterhin wird kein anderes Wort aus der Sprache2 durch f erreicht

Mit einem vergleichbaren Ansatz kann jede entscheidbare Sprache auf jede andere entscheidbare Sprache reduziert werden. Dies zeigt die Idee der Reduktion, hilft aber relativ wenig weiter, da wir Sprache1 $\leq_f$ Sprache2 jeweils gezeigt haben, aber nur jeweils die Aussage, wenn Sprache2 entscheidbar, ist auch Sprache1 entscheidbar, gilt. Spannender ist der typischerweise genutzte Ansatz zu zeigen, dass Sprache1 $\leq_f$ Sprache2 und zu wissen, dass Sprache1 unentscheidbar ist, da dann auch die Unentscheidbarkeit von Sprache2 bewiesen ist.

Frage: Warum wird die Software als Prototyp bezeichnet?

Antwort: Ist genauer ein funktionaler Prototyp, der die (von mir gewünschte) Basisfunktionalität vollständig umsetzt, also komplett nutzbar ist, aber an vielen Stellen Optimierungen benötigt. Dies umfasst zumindest deutlich zu lange Methoden, eine zu geringe Testüberdeckung und die Performance diverser Methoden. Gerade bei der Performance wurde auf die Machbarkeit und eine erste einfache Lösung fokussiert; dies wäre aber im nächsten Schritt zu überarbeiten. Dies ist also kein Wegwerf-Prototyp der für Beispiel-Use-Cases die Machbarkeit zeigt, aber könnte zum wegwerfen animieren, um z. B. den domänengetriebenen Ansatz komplett umzusetzen. Da eine solche Überarbeitung Zeit kostet, ist unklar, wann sie in der Prioritätenliste weiter nach oben wandert.