

Projekt: Entwicklung einer Infsii-Turtle-Code-Programmierungsumgebung

Wir wollen in Anlehnung an Eckart Modrows¹ Projektvorschlag „LOGO für Arme“ im Folgenden in Snap! (oder einer anderen aus dem Unterricht bekannten Programmierungsumgebung) eine textbasierte Bildbeschreibungssprache ähnlich zur Programmiersprache Logo entwickeln. Wir nennen diese selbst entwickelte Sprache *Infsii-Turtle-Code* und wollen mit ihr einfache Turtle-Grafiken erstellen. Hierbei trägt ein virtueller Zeichenroboter, die sogenannte Turtle, einen Stift und kann durch einfache Textbefehle gesteuert werden.

Mögliche Befehle könnten beispielsweise lauten:

Befehl	Bedeutung
F (x)	Die Turtle bewegt sich um x Schritte nach vorne (F orward).
U	Der Stift wird deaktiviert (U p).
D	Der Stift wird aktiviert (D own).
T (x)	Die Turtle dreht sich um den Winkel x im Uhrzeigersinn nach rechts (T urn).

Tabelle 1: Erste Version einer Turtle-Sprache

Mit diesen Textbefehlen erzeugt beispielsweise die folgende Befehlskette die in Abbildung 1 dargestellte Turtle-Grafik:

```
UT(180) F(200) T(270) DF(100) UT(180) F(100) T(90) F(20) T(90) DF(100) T(180) F(100) T(90) F(100) T(135) F(141) T(225) F(100) T(90) UF(20) T(90) DF(100) T(180) F(100) T(90) F(40) T(180) F(40) T(270) F(50) T(270) F(30) T(180) F(30) T(90) F(50) UT(90) F(60) DF(60) T(90) F(100) T(90) F(60) T(90) F(100) T(90)
```

eingabe **UT(180)F(200)T(270)DF(100)UT(180)F(100)T(90)F(20)T(90)DF(100)T(180)F(100)T(135)F(141)T(225)**



Abbildung 1: Beispiel einer Turtle-Grafik

¹ vgl. Modrow, Eckart, Theoretische Informatik mit Delphi, emu-online, 2005 bzw. Modrow, Eckart, „Einführung in die Informatik – Teil VII Erkennende Automaten“, vlin-Material

Ziel des Projektes

Turtle-Code-Umgebungen gibt es bereits in vielfältigen Variationen. Schauen Sie sich beispielsweise einmal die Programmierungsumgebung <https://www.code-your-life.org/turtlecoder/#/user/defaultUser> (Link vom 15.04.2020) an, probieren Sie verschiedene Befehle der Turtle aus und wechseln Sie zwischen grafischem und textbasiertem Modus.

Ziel der Entwicklung einer eigenen *InfSii-Turtle-Code-Programmierungsumgebung* ist die Entwicklung einer textbasierten Programmiersprache, die Konstruktion jeweils eines zugehörigen (vereinfachten) Compilers und Interpreters und damit ein vertieftes Verständnis über deren allgemeine Funktionsweise sowie die Zusammenhänge zwischen formalen Sprachen und Programmiersprachen. Dabei kann die Entwicklung auch arbeitsteilig erfolgen: eine Gruppe kümmert sich um die Implementierung eines Parsers, eine andere um einen Interpreter. Nach Wunsch kann eine dritte Gruppe beispielsweise einen Scanner implementieren, sich jeweils Beispiele zum Testen für die Parser- und die Interpreter-Gruppe überlegen, mögliche Erweiterungen der Befehle diskutieren sowie Grenzen bei der Modellierung der Sprache durch einen endlichen Automaten untersuchen.

Bestandteile einer InfSii-Turtle-Code-Programmierungsumgebung

Definition der Sprache

Zunächst muss die InfSii-Turtle-Sprache eindeutig definiert werden, erst im Anschluss ist ein arbeitsteiliges Vorgehen möglich.

Aufgabe: Falls Sie bereits vorhandene Turtle-Code-Umgebungen getestet haben, haben Sie sicherlich festgestellt, dass dort längere Befehle wie beispielsweise `forward(100)` oder `rightTurn(90)` verwendet werden. Erklären Sie, warum für unsere eigene Sprache bei einer Syntaxanalyse möglichst kurze Befehle vorteilhaft sind.

Um die grundlegenden Prinzipien zu verstehen und frühzeitig bereits eine funktionsfähige Programmierungsumgebung zu erhalten, sollte die InfSii-Turtle zunächst nur über wenige Befehle verfügen (vgl. Tabelle 1, z.B.: Stift aktivieren, Stift deaktivieren, eine Anzahl an Schritten nach vorne bewegen, um eine Anzahl an Grad drehen). Diese können später noch erweitert werden. Sie können die Befehle aus Tabelle 1 verwenden oder sich auf eine eigene Sprachdefinition einigen:

Befehl	Bedeutung
	Die Turtle bewegt sich um x Schritte nach vorne.
	Der Stift wird deaktiviert.
	Der Stift wird aktiviert.
	Die Turtle dreht sich um den Winkel x im Uhrzeigersinn nach rechts.
	... (werden später ergänzt)



Scanner

Soll die Eingabe zur besseren Lesbarkeit Leerzeichen, Tabulatoren sowie eine Mischung von Groß- und Kleinschreibung erlauben, so bereitet der Scanner diese für die weitere Untersuchung des Parsers vor. Dabei werden beispielsweise Leerzeichen entfernt. In Snap! wird beim Vergleich von Zeichenketten nicht zwischen Groß- und Kleinschreibung unterschieden. Wird die `InfSii-Turtle-Code`-Programmierungsumgebung in einer anderen Programmiersprache implementiert, muss der Scanner möglicherweise noch alle Befehle in Großbuchstaben (bzw. abhängig von der Wahl der Befehle in Kleinbuchstaben) übersetzen.

Bei der Entwicklung einer eigenen Programmierungsumgebung kann die Realisierung eines Scanners entfallen. Dann werden Eingaben mit Leerzeichen, Mischung von Groß- und Kleinschreibung usw. vom Parser als ungültig interpretiert.

Parser

Beim Kompilieren eines Quelltextes wird u.a. überprüft, ob dieser syntaktisch korrekt ist. Diese Überprüfung einer Eingabe (d.h. des Quelltextes) kann mithilfe eines endlichen Automaten erfolgen. Abhängig von der Art der Eingabe gibt der Parser unterschiedliche Rückmeldungen. Entscheiden Sie bei der Umsetzung, wie detailliert diese Rückmeldungen sein sollen. Wird nur entschieden, ob die Eingabe syntaktisch korrekt oder nicht korrekt war? Oder sollen qualifiziertere Rückmeldungen erfolgen, wie beispielsweise „Fehler: Ziffer erwartet, stattdessen D gelesen“ oder „Fehler: Eingabe unvollständig“?

Sie entscheiden selbst, wie umfangreich Ihr Parser gestaltet sein soll. Die Mindestanforderung ist, dass syntaktisch falsche Eingaben erkannt werden. Wenn Sie sich auf eine konkrete Sprachdefinition geeinigt haben, können Sie eigenständig mit der Entwicklung eines Parsers (d.h. Erstellung eines geeigneten Automatenmodells sowie zugehörige Implementierung in einer aus dem Unterricht bekannten Programmiersprache) beginnen. Alternativ können Sie sich an den InfSii-Materialien zur Entwicklung eines Parsers in Snap! orientieren.

Interpreter

Wir entwickeln unsere `InfSii-Turtle-Code`-Programmierungsumgebung mithilfe von Snap! oder einer anderen aus dem Unterricht bekannten Programmiersprache. Das bedeutet, dass unser Turtle-Code-Interpreter den InfSii-Turtle-Code in diese Programmiersprache übersetzen muss, der darin enthaltene Interpreter übernimmt dann die weitere Ausführung. Dabei kann der Turtle-Code-Interpreter davon ausgehen, dass der Parser syntaktisch falsche Eingaben erkannt hat und nur zur Sprache gehörende Ausdrücke weitergegeben wurden. Wenn Sie sich auf eine konkrete Sprachdefinition geeinigt haben und von syntaktisch korrekten Ausdrücken ausgehen, können Sie eigenständig mit der Entwicklung eines Interpreters beginnen. Unter Umständen hilft auch hier eine Modellierung mit einem geeigneten Automatenmodell (z.B. einem Mealy-Automaten). Alternativ können Sie sich an den InfSii-Materialien zur Entwicklung eines Interpreters in Snap! orientieren.

Mögliche Erweiterungen der Sprachdefinition

Die Infsii-Turtle hat zunächst nur wenige Befehle. In der folgenden Tabelle werden weitere mögliche Befehle vorgestellt. Diese Liste ist keineswegs vollständig und kann um eigene Ideen ergänzt werden.

Aufgabe: Wählen Sie mehrere neue Befehle aus und überlegen sich geeignete Codierungen. Erweitern Sie anschließend Ihren Parser und Compiler entsprechend.

Befehl	Bedeutung
	Die Turtle bewegt sich um x Schritte rückwärts.
	Die Turtle dreht sich um den Winkel x entgegen dem Uhrzeigersinn.
	Die Stiftdicke wird geändert.
	Die Stiftfarbe wird geändert.
	Die Stiftfarbe wird auf den RGB-Wert (r, g, b) gesetzt.
	Die Turtle versteckt sich.
	Die Turtle wird sichtbar.
	Der Bildschirm wird gelöscht.
	Die Turtle geht zur Position (x, y) .
	Eine Befehlsfolge wird x-mal wiederholt.

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#). Sie erlaubt Bearbeitungen und Weiterverteilung des Werks unter Nennung meines Namens und unter gleichen Bedingungen, jedoch keinerlei kommerzielle Nutzung.