Matthias Heming 25. November 2009

Hinweise zur Phase »Sprachen«

1 Kompetenzformulierung

Schülerinnen und Schüler ...

- erfassen die Beschränktheit von Rechenmaschinen anhand der begrenzten Anzahl und Einfachheit verfügbarer Befehle.
- unterscheiden Maschinensprache und (höhere) Programmiersprache und grenzen diese von Fach- und Umgangssprache ab.
- erkennen Compiler bzw. Interpreter als automatische Übersetzungseinheit.
- erfassen den Begriff der *Programmierung* als Übersetzungstätigkeit zwischen Fachsprache und Programmiersprache, die nicht/nur schwer automatisiert werden kann.
- lesen und nutzen Railroad-Diagramme zur Darstellung sprachlicher Regeln (Grammatiken).
- unterscheiden Terminal- und Nonterminal-Symbole.

2 Detaillierte Zielsetzung

Zur Abgrenzung der unterschiedlichen Sprachdefinitionen (Maschinensprache, Assemblersprache, Programmiersprache, höhere Programmiersprache, Systemsprache, Fachsprache, Umgangssprache) geht diese Phase auf die geschichtliche Entwicklung der Computer/Rechenmaschinen ein und erläutert die unterschiedlichen Schnittstellen, mit denen der Mensch eine Maschine manipulieren kann. Dabei muss darauf geachtet werden, dass die Begriffsbildung, die gerade bei der Gegenüberstellung von Rechenmaschinen/Computern und Informatiksystemen schwer zu erfassen ist, in sich konsistent bleibt.
Die erste Aufgabenstellung kann inhaltlich durch Nachschlagen des entsprechenden Wikipedia-Artikels[1] gelöst werden. Es kann also sowohl der Artikel – oder ein Ausschnitt davon – ausgedruckt ausgeteilt werden, als auch als Rechercheauftrag innerhalb einer geeigneten Infrastruktur formuliert werden. Als stark vereinfachtes Bildes wird die Zuse Z3 hier als einfache Rechenmaschine bezeichnet, die nicht selbstständig arbeiten kann. Sie stellt nur Hardware dar, welche durch entsprechende Befehle in Form von Lochkarten (Software) gesteuert werden muss. Auch wenn dies nicht exakt die Realität widerspiegelt[2], so erleichtert diese Analogie zumindest das Verständnis für den Begriff des Informatiksystems. Es wird wiederholt, dass ein Informatiksystem im Sinne der Definition zwingen einen Softwareanteil beinhaltet.
Reduziert man Assemblersprachen auf eine 1:1 Übersetzung von Befehlen (in Form von Zeichenketten) in die dazu passende (binäre) Kodierung[3], so können die beiden unterschiedlichen Formen als Maschinencode bzw. Maschinensprache

[1] `http://de.wikipedia.org/wiki/Zuse_Z3#Betrieb` – geprüft am 07. Oktober 2009
[2] Das Rechenwerk des Z3 kann als eine Art Mikroprogramm – und damit Software – angesehen werden.
[3] Dies ist nicht ganz korrekt, da die Übersetzung des Assemblercodes in den Maschinencode auch die Umrechnung symbolischer Adressen in absolute beinhaltet und Kommentare wegfallen, wodurch der Prozess nicht mehr exakt umkehrbar wird.

bezeichnet werden. Auf dieser theoretischen Ebene sind also *Assemblercode* und Maschinen*sprache* gleichzusetzen und von Maschinen*code* abzugrenzen. Für Schülerinnen und Schüler in dieser Phase sollte der Begriff der Maschinensprache bereits komplex genug sein, falls dennoch nachfragen auftreten, so können die Begrifflichkeiten konsistent den vorliegenden Begriffsrahmen eingebettet werden.
Prinzipiell kann die Maschinensprache bereits als Programmiersprache bezeichnet werden, jedoch ist die praktische Programmierung mit einer solchen Sprache sehr mühsam. Die Bezeichnung der höheren Programmiersprache wird daher auf das höhere Abstraktionsniveau dieser Sprachen zurückgeführt. Auch wenn Assemblersprachen durch symbolische Adressierung und Umwandlung in Mnemonics[4] ebenfalls in einem gewissen Grad abstrahieren, geschieht dieses auf einem völlig anderen Niveau, als es Programmiersprachen wie C, C++, Java oder Python durchführen. Die nächste Aufgabe soll darauf hinführen, dass die Erfindung solcher Sprachen nur deswegen praktisch sinnvoll ist, da automatische Mechanismen existieren, die den Quelltext in einer höheren Programmiersprache auf die Ebene der Maschinensprache bzw. des Maschinencodes überführen können (Compiler/Interpreter).
Als letztes muss der Sprung zwischen einer Programmiersprache und der Fachsprache bzw. Umgangssprache geschafft werden. Während die Fachsprache mit dem Ziel der klaren und exakten Kommunikation durch entsprechende Definitionen bereits ansatzweise formalisiert wurde, bieten Railroad-Diagramme einen noch stärkeren formalen Zugang zu einzelnen Sprachelementen, der durch die visuelle Darstellung ein wenig vereinfacht wird.
Man kann schnell erkennen, dass die deutsche Sprache sehr flexibel gestaltet ist und sich daher nur schwer in solch formale Strukturen zwängen lässt, wie sie von Railroad-Diagrammen festgelegt sind. Dabei wird im Gegenzug die Einfachheit einer Programmiersprache gezeigt, wodurch eine positive Selbsteinschätzung erzeugt werden soll (»deutsch spreche ich flüssig, dann wird so eine einfache Sprache doch leicht erlernbar sein«).

3 Mögliche Weiterarbeit

- Die Darstellung verschiedener sprachlicher Strukturen durch Railroad-Diagramme kann (evtl. in Zusammenarbeit mit dem Deutschunterricht) vertieft werden.
- Formulierung etwas komplexerer Regeln der Programmiersprache Python. Zur Aufgabe zum Fließkommazahl-Datentyp könnte hinzugefügt werden, dass eine Angabe einer Zehnerpotenz möglich ist (Beispiel: `1234.98e10`). Arithmetische Ausdrücke, Listen- oder Dictionary-Eingaben evtl. unter Benutzung von Bezeichnern...
- Analyse automatisch übersetzter Texte: Woran scheitern Informatiksysteme? Was ist *Übersetzunugsgerechtes Schreiben*[5]? Dies kann in Zusammenarbeit mit fremdsprachlichen Fächern getan werden.
- Werden Informatiksysteme die menschliche Sprache beherrschen (können)? Mit Bezugnahme zu Eliza und dem Turingtest.
- Die formale Sprache von Pfadnamen für Dateien und Verzeichnisse.

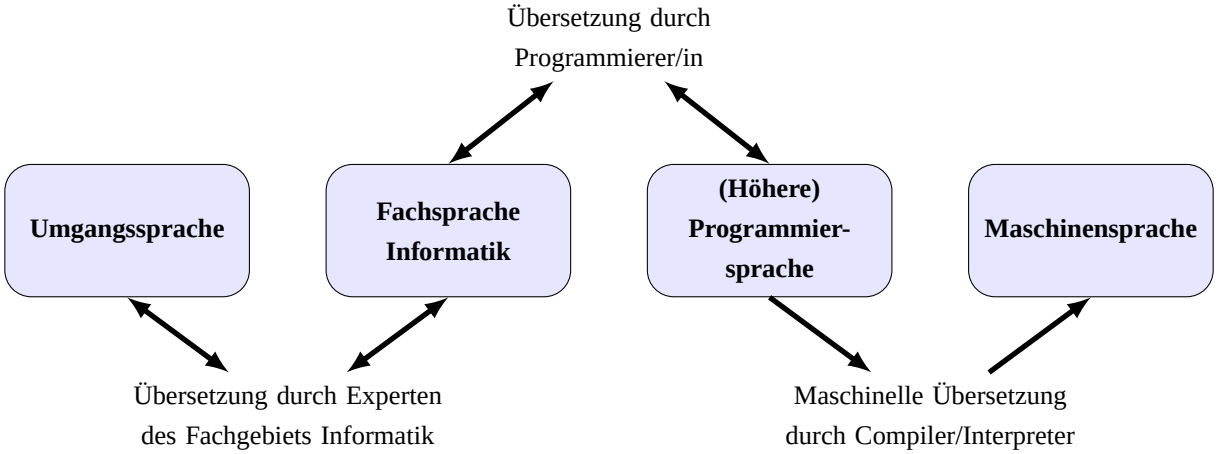
4 Genderaspekt

Da sich Mädchen eher sprachliche Fähigkeiten zutrauen (Achtung: Stereotyp!), wird durch die Definition des Programmierens als übersetzende Tätigkeit, der Fokus auf die Programmierung mit einer Programmier*sprache* gelenkt. Dies beinhaltet die Hoffnung, dass sich damit das Selbstvertrauen der Mädchen erhöht.

[4] Es wird z. B. dem Maschinencode für einen Additionsbefehl mit `add` ein Name (Mnemonic) gegeben.
[5] Siehe z. B. `http://de.wikipedia.org/wiki/Übersetzungsgerechtes_Schreiben` – zuletzt geprüft am 08. Oktober 2009

5 Für das Merkheft

Die folgende Grafik gibt eine Übersicht über verschiedene Sprachtypen und die Übersetzungsmöglichkeiten dazwischen.



Programmieren: Das Übersetzen eines fachsprachlich formulierten Algorithmus in eine höhere Programmiersprache. Das Ergebnis nennt man Quelltext.

Theoretisch kann der Algorithmus auch direkt in der Maschinensprache geschrieben werden, dass ist jedoch sehr umständlich, da man sich um viele technische Details kümmern muss.

Interpreter: Ein fertiges Programm, dass Befehle einer bestimmten (höheren) Programmiersprache einliest und die Befehle direkt in die Maschinensprache übersetzt und ausführt.

Compiler: Ein fertiges Programm, dass Befehle einer bestimmten (höheren) Programmiersprache einliest und in die Maschinensprache übersetzt. Das Ergebnis kann zu einem gewünschten späteren Zeitpunkt ausgeführt werden.

Begriffe im Zusammenhang mit Railroad-Diagrammen:

Terminal(symbol): Ein Symbol einer (formalen) Sprache. Es kann nicht weiter in Einzelteile zerlegt werden.

Nonterminal(symbol): Eine Variable, die durch bestimmte Regeln noch in mehrere oder ein einzelnes Terminalsymbol umgewandelt werden muss. In dem späteren Satz kommt das Nonterminalsymbol nicht vor, es wird nur zum Zusammenbau des Satzes benutzt.