Ein Intelligentes Tutorensystem für Rekursive Programmierung

Fehlkonzepte identifizieren für zielgerichtetes Feedback

Sonja Niemann, Anna M. Thaler, Ute Schmid

Abstract: *Im Beitrag wird ein Intelligentes Tutorensystem (ITS) vorgestellt, das Programmierfehlern zugrundeliegende Fehlkonzepte identifizieren kann. Das hier vorgestellte ITS ist für den Bereich der rekursiven Programmierung in Python entwickelt. Kern des ITS ist eine Methode, bei der diagnostische Tests auf dem von Studierenden eingegebenen Programmcode durchgeführt werden. Fehlerhafte Testfälle sind spezifischen Fehlkonzepten, die für diesen Fehler ursächlich sind, zugeordnet. Dabei wird eine in der Informatikdidaktik entwickelte Taxonomie von Fehlkonzepten beim rekursiven Programmieren genutzt (Hamouda et al., 2017). In einem nächsten Schritt werden die identifizierten Fehlkonzepte genutzt, um Rückmeldungen zu generieren, die so spezifisch und so individuell wie möglich auf die Lernenden abgestimmt sind.*

In this contribution, we present an intelligent tutor system (ITS) that can identify misconceptions based on programming errors. The ITS has been developed to teach recursive programming in Python. Core of the ITS is a method based on diagnostic tests for program code of students. Erroneous test cases are mapped to specific misconceptions underlying the observed errors. The mapping is based on a taxonomy of typical misconceptions of recursion identified in programming education (Hamouda et al., 2017). In a next step, the identification of misconceptions can be used to tailor the feedback as precisely and individually as possible to the students.

Keywords: *Intelligentes Tutorensystem; Rekursion; Python; Fehlkonzepte; Testbasierte Diagnose; individualisierte Rückmeldungen; intelligent tutor system; recursion; python; misconceptions; test-based diagnosis; targeted feedback*

1. Einleitung

Zu Beginn jedes Informatikstudiums müssen grundlegende Programmierfertigkeiten erlernt werden. Eine dieser Fertigkeiten ist das rekursive Lösen von Problemen. Dies stellt jedoch oft eine große Herausforderung für Programmieranfängerinnen und -anfänger dar. Es gibt zahlreiche Untersuchungen dazu, welche Probleme Studierende haben,wenn sie lernen Probleme rekursiv zu lösen (Götschi et al., 2003; Hamouda et al., 2017; Sanders & Scholtz, 2012). Diese Erkenntnisse können von Tutorinnen und Tutoren aber nur dann genutzt werden, wenn sie genug Zeit haben, um alle Studierenden individuell zu betreuen. Um die Lehrpersonen zu entlasten, wurde ein Intelligentes Tutorensystem (ITS) entwickelt, das den Studierenden helfen kann, rekursive Funktionen zu programmieren. ITS sind seit den 1970ern Bestandteil der Forschung von KI in der Lehre (Carbonell,1970). Sie bestehen klassisch aus vier Modulen: dem Experten/Domänen-Modul, dem Pädagogisches/Tutor-Modul, dem Studierenden-Modul und dem Interface-Modul (Nwana, 1990; Raza, 2020; Rus et al., 2013). Der Fokus des vorgestellten ITS liegt darauf, dass die Studierenden ihre Funktionen selbst programmieren und im Vergleich zu bestehenden Programmier-ITS keine Lücken füllen oder Multiple-Choice-Aufgaben lösen müssen. Es werden keine Syntaxfehler korrigiert, sondern zugrundeliegende Fehlkonzepte identifiziert, damit Studierende konkret Hilfe für die Bereiche bekommen können, die sie noch nicht richtig verstanden haben. Die Nutzung setzt voraus, dass Studierende bereits grundlegende Kenntnisse in Python sowie Rekursionen haben.

2. Intelligente Tutorensysteme und Rekursion

2.1 Rekursion

Rekursion spielt in vielen Disziplinen eine entscheidende Rolle. So betrachtet man in der *Theory of Mind* die Fähigkeit zum rekursiven Denken. Es geht darum, wann Menschen in ihrer Entwicklung die Fähigkeit erlernen, sich in andere Menschen hineinzuversetzen und Annahmen darüber zu machen, was diese denken (Valle et al., 2015). In der Informatik werden rekursive Funktionen dadurch beschrieben, dass sie sich selbst mit einem kleineren Teilproblem aufrufen. Die Aufrufe werden so lange ausgeführt, bis ein definierter Basecase erreicht ist (Becker, 2023). Als Anfangsaufgabe in dem ITS wurde das Berechnen einer Summenfolge von einer Zahl n gewählt. Dabei wird n mit all seinen Vorgängern aufsummiert. Für n = 4 würde dies so berechnen: $4 + 3 + 2 + 1 + 0 = 10$. Der Rekursive Ansatz sieht vor, dass die Funktion sum(n) sich selbst aufruft, aber n mit jedem Aufruf verkleinert. Die Summenfolge von 4 ist nicht bekannt, daher wird das Problem vereinfacht und als 4 + sum(3) dargestellt. Das Problem wird so lange vereinfacht, bis der definierte

Basecase erreicht ist, in diesem Beispiel n = 0. Für den Fall von n = 0 ist das Ergebnis definiert, dieser wird nun im passiven flow an die Aufrufe zurückgegeben, sodass wir rückwirkend alle Summen berechnen können.

2.2 Fehlkonzepte

Ziel des ITS ist es Fehlkonzepte zu identifizieren, dazu müssen diese zunächst definiert werden. Welche Probleme und Schwierigkeiten Studierende beim Erlernen von Rekursionen haben, wurde bereits ausführlich erforscht (Götschi et al., 2003; Hamouda et al., 2017; Sanders & Scholtz, 2012). Um das ITS so zu modellieren, dass es identifizieren kann auf welche Aspekte Programmierfehler genau zurückzuführen sind, ist eine Art Kategorisierung notwendig. Hamouda et al. (2017) haben typische Fehler von Studierenden zugrundeliegenden Fehlkonzepten zugeordnet. Sie definieren Fehlkonzepte als eine falsche Idee oder Annahme, die auf dem Missverständnis von etwas beruht. Das Ziel des ITS ist es genau dieses grundliegende Konzept zu finden, das missverstanden wurde, z.B. welche Fehler passieren, wenn Studierende nicht genau verstanden haben, wie der aktive Flow abläuft. Hamouda et al. (2017) haben 5 Kategorien identifiziert, in denen 12 Fehlkonzepte definiert wurden. Dadurch kann genau beschrieben werden, was Studierende nicht verstanden haben. Das resultierende Concept Inventory beinhaltet Aufgaben, sowie ergänzend typische falsche Antworten von Studierenden, die den zugrundeliegenden Fehlkonzepten zugeordnet werden (Hamouda et al., 2017). Diese Form von Zuordnung von falschen Lösungen zu Fehlkonzepten hat die Überlegung angestoßen ein ähnliches Vorgehen zu wählen, bei dem Studierende jedoch selbst Programmieren können.

3. Testpipeline

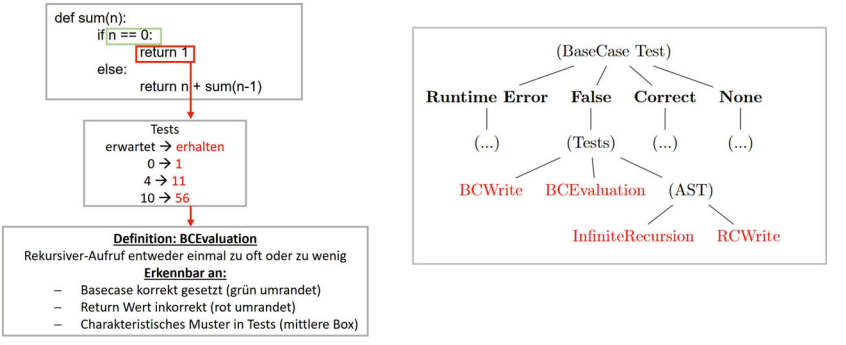
3.1 Mapping

Das Vorgehen von Hamouda et al. (2017) musste auf die neue Aufgabenstellung angepasst werden. Die Aufgaben aus dem Concept Inventory eignen sich leider nicht als Programmieraufgaben, da sie in keine erkennbare logische Funktion haben, also nur als Tracing-Aufgaben verwendbar sind. Da sich das ITS an Programmieranfängerinnen und Anfängern richtet wurden passende Aufgaben von Inf-Schule (»inf-schule.de«, 2023) ausgewählt, die Seite wurde von Informatik-Lehrenden gestaltet. Die beschriebenen Fehlkonzepte aus dem Concept Inventory (Hamouda et al., 2017) wurden dazu herangezogen, um eine Fehlerdatenbank zu erstellen.

In Abb. 1 ist ein Beispiel für das Fehlkonzept BCEvaluation, das Fehlkonzept zeichnet sich dadurch aus, dass die Ergebnisse jeweils um 1 zu groß oder zu klein sind. Hamouda et al. (2017) führen dies darauf zurück, dass Studierende nicht

verstanden haben, wie oft der rekursive Aufruf durchgeführt wird. Um ein Fehlkonzept zu identifizieren, müssen mehrere Tests durchgeführt werden, bis ein charakteristisches Muster gefunden wurde. Der Auszug aus der Fehlertaxonomie in Abb. 1. verdeutlicht, wie verzweigt die Testpipeline sein muss um möglichst viele Fehlkonzepte abzudecken. Mit diesem Vorgehen können bereits 7 der 12 im Concept Inventory definierten Fehlkonzepte identifiziert werden.

Abb. 1: Mappingprozess anhand einer Beispielaufgabe (links) mit Auszug aus der Fehlertaxonomie (rechts).



3.2 Testarten

Das ITS verwendet zwei verschiedene Testarten um möglichst viele Fehlkonzepte identifizieren zu können. Wie in Abb. 1 dargestellt, besteht die erste Option darin, einfache Testwerte zur Überprüfung zu verwenden. Für jede Aufgabe wird der Basecase sowie drei weitere Inputvariablen hinterlegt. Können mehrere Fehlkonzepte für eine Fehlermeldung verantwortlich sein ist es notwendig eine weitere Testform hinzuzuziehen. Um eine genaue Unterscheidung möglich zu machen, kann das ITS die Abstract Syntax Trees (AST) des Studierenden Codes auf bestimmte Fehlkonzepte untersuchen. AST ist das Zwischenprodukt des Compilers, bevor der Code in maschinenlesbare Form übersetzt wird. Zeichen werden dabei zu Tokens gruppiert, die einen logischen Zusammenhang haben, Zeichen die einen rein syntaktischen Nutzen haben, sind nicht mehr repräsentiert (Noonan, 1985). Liegt der Code in dieser Form vor, kann nach verallgemeinerten Operationen gefiltert werden. Anstatt nach dem Ausdruck *sum(n-1)* zu suchen, der nur für die Aufgabe sum() gilt, können ASTs allgemein nach rekursiven Aufrufen durchsucht werden. In einem zweiten Schritt wird zum Beispiel überprüft, ob die Inputvariablen reduziert werden, also ob *(n-1)* geschrieben wurde.

4. Implementierung

Studierende können sich mit einem Nutzernamen und Passwort bei der Webapplikation anmelden und ein Profil erstellen, in dem auch ihr Lernfortschritt gespeichert werden. Es wurden 6 Aufgaben in aufsteigender Schwierigkeit mit dazugehöriger Lösung und Angaben für die Testpipeline in dem ITS integriert. Alle Informationen zu Aufgaben, Studierenden und ihren Lösungen sind in einer Datenbank gespeichert. Wurde ein Fehlkonzept identifiziert haben Studierende die Möglichkeit die Aufgabe erneut zu lösen, jedoch nicht mit der nächsten Aufgabe weiterzumachen. Die nächstschwierigere Aufgabe wird erst angezeigt, wenn die voran gegangene richtig gelöst wurde.

5. Zusammenfassung und Ausblick

Die Grundlage für hilfreiches Feedback wurde mit dem aktuellen Stand des ITSs gegeben. Das Feedback Modul sollte in einem nächsten Schritt dahin erweitert werden, dass die Information über das identifizierte Fehlkonzept genutzt wird, um Studierende bestmöglich zu unterstützen. Es ist denkbar strukturanaloge Beispiele zu generieren, oder den Ablauf der Rekursion zu visualisieren, damit Studierende besser erkennen, an welcher Stelle sie einen Fehler machen. Ist das Feedback Modul ausgereift müsste eine experimentelle Studie zeigen, inwieweit Studierenden vom dem ITS profitieren.
Falsche Lösungen von Studierenden zu Fehlkonzepten zuzuordnen könnte eine interessante Herangehensweise für weitere schwer formalisierbare Konzepte sein, setzt jedoch eine ausführliche Ausarbeitung der Fehlkonzepte voraus.

Förderhinweis: Der Beitrag ist im Rahmen des Projekts »Digitale Kulturen der Lehre entwickeln« entstanden, gefördert durch die *Stiftung Innovation in der Hochschullehre*.

Literaturverzeichnis

Carbonell, J. R. (1970). AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems*, *11*(4), 190–202. https://doi.org/10.1109/TMMS. 1970.299942

Götschi, T., Sanders, I., & Galpin, V. (2003). Mental Models of Recursion. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 346–350.

Hamouda, S., Edwards, S. H., Elmongui, H. G., Ernst, J. V., & Shaffer, C. A. (2017). A Basic Recursion Concept Inventory. *Computer Science Education*, *27*(2), 121–148.

Noonan, R. E. (1985). An Algorithm for Generating Abstract Syntax Trees. *Computer Languages*, *10*(3–4), 225–236.

Nwana, H. (1990). *Intelligent Tutoring Systems: An Overview*.

Station – Rekursion. (2023, 05. Oktober). *inf-schule. https://www.inf-schule.de/* (Abgerufen am 28.04.2024)

Raza, A. (2020). Intelligent Tutoring Systems and Metacognitive Learning Strategies: A Survey. *Muhammad ZAYYAD*, 47.

Rus, V., D'Mello, S., Hu, X., & Graesser, A. (2013). Recent Advances in Conversational Intelligent Tutoring Systems. *AI Magazine*, *34*(3), 42–54.

Sanders, I., & Scholtz, T. (2012). First Year Students' Understanding of the Flow of Control in Recursive Algorithms. *African Journal of Research in Mathematics, Science and Technology Education*, *16*(3), 348–362.

Valle, A., Massaro, D., Castelli, I., & Marchetti, A. (2015). Theory of Mind Development in Adolescence and Early Adulthood: The Growing Complexity of Recursive Thinking Ability. *Europe's Journal of Psychology*, *11*(1), 112.