Stand des DFG-Projekts "Entwicklung einer Wissensdiagnostik- und Fehlererklärungskomponente beim Erwerb von Programmierwissen für ABSYNT"

Hans Colonius, Klaus-Dieter Frank, Klaus Kohnert,
Gabriele Meyer, Claus Möbus,
Olaf Schröder, Heinz-Jürgen Thole
Universität Oldenburg
Fachbereich Informatik

Ziel des Projekts ist die Entwicklung eines Wissensdiagnose- und Fehlererklärungssystems beim Erlernen des funktionalen Programmierens in ABSYNT; einer graphischen Programmiersprache, die eine Erweiterung von "abstract syntax trees" darstellt.
Die Wissensdiagnostikkomponente soll den jeweils aktuellen Wissensstand des Lernenden (syntaktisches, Berechnungs- und Planungswissen) repräsentieren. Fehler sollen auf falsches oder fehlendes Wissen zurückgeführt werden.
Die Fehlererklärungskomponente soll auf der Basis der von der Wissensdiagnostikkomponente gelieferten Information und unter Berücksichtigung des individuellen Lernerverhaltens angemessene Hilfestellungen geben.

1. Die Programmiersprache ABSYNT

Mit ABSYNT sollen sprachübergreifende Programmierkonzepte, die für funktionale Sprachen typisch sind, mit einfachen Datentypen erlernt werden. Die Vermittlung sprachübergreifender Konzepte scheint geeignet, den Lernenden auf späteres, komplexes Programmieren vorzubereiten (Neber, 1985) und den Transfer zu anderen, nicht graphischen, funktionalen Programmiersprachen zu erleichtern (Möbus, 1985).
ABSYNT wird im Projekt aus nichtformalen Illustrationen (Schmitt & Wohlfarth, 1978; Bauer & Goos, 1982) zu einer lauffähigen Programmiersprache weiterentwickelt und auf

einem Arbeitsplatzrechner in Interlisp und LOOPS implementiert. Ihre Auswahl und Weiterentwicklung ist durch folgende Eigenschaften motiviert:

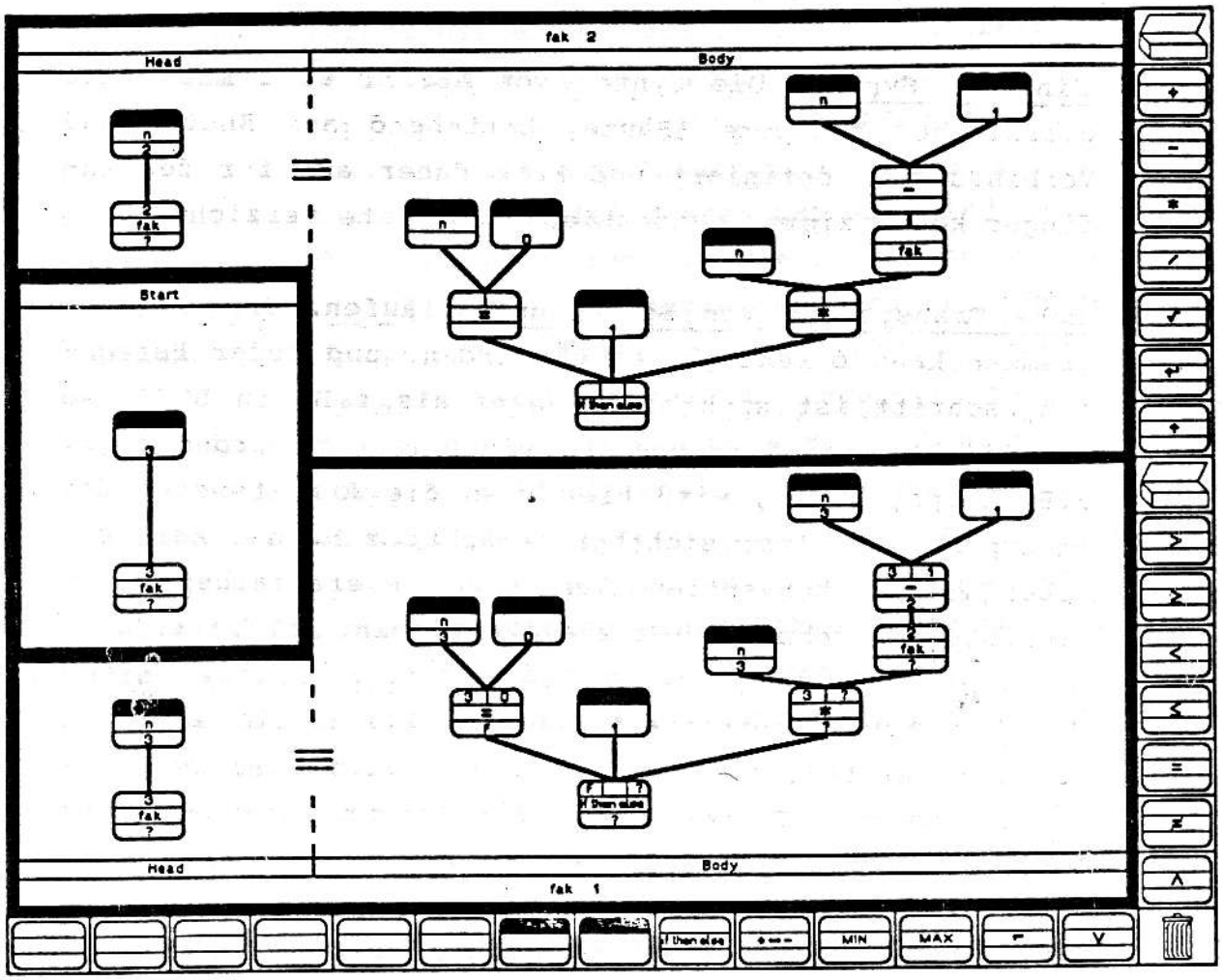
- Einfache Syntax. Die Syntax von ABSYNT wird mit Hilfe graphischer Konzepte (Bäume, bestehend aus Knoten und Verbindungen) definiert und kann daher auf für den Anfänger komplizierte syntaktische Konzepte verzichten.

- Hohe Transparenz von Berechnungsabläufen. In den Programmen kann direkt gerechnet werden, und jeder Berechnungsschritt ist sichtbar. Anders als z.B. in LOGO, wo die Effekte von Berechnungen visualisiert werden (vgl. z.B. Hoppe, 1985), wird hier also die Arbeitsweise des Interpreters selbst sichtbar gemacht; z.B. der Auf- und Abbau von in konventionellen Programmiersprachen nicht sichtbaren Umgebungen (Parameter-Argument-Bindungen).

- Direkte Manipulierbarkeit von Objekten. Die Elemente eines zu erstellenden Programms können mit Hilfe eines mausgesteuerten Cursors einer Menüleiste entnommen oder auf dem Bildschirm bewegt werden. Der Lernende kann seine Ideen schnell umsetzen und extern bearbeiten (Hollan et al., 1986; Shneiderman, 1986).

Die Abbildung zeigt ein Stadium in der Berechnung eines ABSYNT-Programms für die Fakultät mit dem Wert 3. Jeder Knoten enthält ein Eingabefeld, ein Namensfeld und ein Ausgabefeld. (Bei Konstanten sind Namens- und Ausgabefeld identisch und daher optisch nicht differenziert). Die Fragezeichen symbolisieren Berechnungsziele.

Abbildung: Ein Stadium in der Berechnung des ABSYNT-Programms "fak" (Fakultät) mit dem Wert 3.



Die Berechnung (call-by-value-Semantik mit paralleler Parameter-Argument-Bindung) beginnt mit dem Setzen eines Berechnungsziels im Ausgabefeld der Wurzel des Startbaums. Das Berechnungsziel wandert in das Eingabefeld und wird sodann durch den Wert der Konstanten im Startbaum ersetzt. Nun wird die Rechenvorschrift "fak" aufgerufen. Dazu wird ein Rahmen mit der Inkarnationsnummer 1 angelegt. Der Parameter des Kopfes dieses Rahmens wird gebunden; das Berechnungsziel wandert in die Wurzel des Kopfes und anschließend weiter in die Wurzel des Körpers. In der Abbildung ist gerade der Rahmen (mit der Inkarnationsnummer 2) für den ersten rekursiven Aufruf angelegt worden.
Der Rahmen, in dem gerade gerechnet wird, befindet sich stets in der oberen Bildschirmhälfte. Die untere Bildschirmhälfte enthält den vorhergehenden Rahmen, so daß die Aufrufstelle sichtbar bleibt. Es werden also Rahmen gekellert. Ist die Berechnung eines Rahmens beendet (die Wurzel seines Kopfes trägt einen Ausgabewert), so wird das Ergebnis in die Aufrufstelle übertragen, und der Rahmen wird gelöscht.
Die Semantik von ABSYNT ist durch ein Regelsystem in PROLOG festgelegt. Dies erscheint für unsere Projektzwecke angemessen, weil das Berechnungswissen Lernender ebenfalls auf Regelebene modelliert werden wird, und weil ein Regelsystem als günstige Ausgangsbasis für die Formulierung von Berechnungsinstruktionen erscheint (s.u.).
Das Regelsystem legt die Darstellungsform der Programme (z.B. die Dreiteilung der Knoten; den Funktionsrahmen) fest. Alle zur Berechnung notwendigen Informationen sind auf dem Bildschirm sichtbar. Die Darstellungsform ist ferner durch die Ergebnisse einer explorativen Untersuchung motiviert, die weiter unten beschrieben wird.
Der rechte und der untere Rand der Abbildung zeigen die Menüleiste für die Auswahl von Knoten beim Programmieren. Verbindungslinien können durch Klicken der Maustaste erzeugt werden. Knoten und Verbindungen können ferner über

entsprechende Menüs verschoben bzw. gelöscht werden.

2. Zum Programmierwissen für die Programmiersprache ABSYNT

Wir betrachten das Programmieren als eine Form des Problemlösens, die auf verschiedenen ineinandergreifenden Wissensquellen basiert.
Planungswissen kann z.B. als Hierarchie von Lösungsplänen auf verschiedenen Abstraktionsebenen (strategisch, taktisch, implementierend) aufgefaßt werden (Soloway et al., 1984). Lösungspläne können dem geübten Programmierer als "canned solutions" (Soloway, 1986) fertig zur Verfügung stehen, oder sie müssen mit Hilfe verschiedener Heuristiken (z.B. Analogiebildungen; Anderson et al., 1984) entwickelt werden. Semantisches Wissen ist bereits auf der Planungsebene sowie bei der Implementation von Plänen (Brooks, 1977, 1983) relevant; hierbei kommt auch syntaktisches Wissen ins Spiel. Semantische Unklarheiten des Lernenden scheinen eine wesentliche Quelle von Fehlern beim Programmieren zu sein (du Boulay & O'Shea, 1981).
Für die Entwicklung einer Wissensdiagnostikkomponente für den vorliegenden Gegenstandsbereich sind die Repräsentationen korrekten Wissens ("Expertenmodell") und fehlerhaften oder unvollständigen Wissens ("Schülermodell") auf allen genannten Ebenen notwendig. Die Unterschiede zwischen Experten- und Schülermodell ergeben die diagnostischen Hypothesen über Fehlkonzepte des Lernenden hinsichtlich Syntax, Semantik und Planung.
Der Schwerpunkt unserer Arbeit liegt gegenwärtig auf dem semantischen Wissen.

2.1 Explorationen zu syntaktischem und semantischem Wissen

Es wurde eine explorative Untersuchung anhand der in Bauer & Goos (1982) eingeführten "Formularsprache", auf die sich ABSYNT stützt, mit folgenden Zielsetzungen

durchgeführt:

- Exploration folgender Fragen: Kommen Lernende mit dieser Programmiersprache zurecht? Wie werden die Programme bearbeitet und repräsentiert?
- Kriterien für das Design der Darstellungsform von ABSNYT.
- Erhebung syntaktischer und semantischer Fehler als notwendiger Vorbereitungsschritt für die Entwicklung der Wissensdiagnostikkomponente.

In der Untersuchung wurden Programme der Formularsprache in verschiedenen Schwierigkeitsstufen (z.B. Programme mit Fallunterscheidungen und Abstützungen; rekursive Programme) von vorher eingearbeiteten Programmiernovizen eingeprägt bzw. anhand verbal formulierter Instruktionen berechnet und anschließend reproduziert oder mit Vergleichsprogrammen aus dem Gedächtnis auf Wertverlaufsgleichheit beurteilt.

Repräsentation der Programme. Die Vergleichsprogramme waren so konstruiert, daß sie zwischen hypothetischen syntaktischen und semantischen Repräsentationen der vorher bearbeiteten Programme trennten.

Die Ergebnisse legten nahe, daß eingeprägte Programme anhand syntaktischer Merkmale (meist nach syntaktischen Kriterien gruppierte Beschriftungselemente) repräsentiert werden. Dies steht im Einklang mit anderen Untersuchungen (z.B. Hoc, 1977; Adelson, 1981), die gezeigt haben, daß Programmiernovizen in hohem Maße syntaxgebunden vorgehen, solange es die Aufgabenstellung zuzulassen scheint.

Berechnete Programme hingegen werden semantisch repräsentiert: als funktionale Abstraktionen (Dosch, 1984) oder als Eingabe-Ausgabe-Wertebeispiele. Zusätzlich wurden in erhobenen Verbalprotokollen ähnliche funktionale Beschreibungen auch der Binnenstruktur der berechneten Programme beobachtet (vgl. Brooks, 1983; Letovsky, 1986).

Syntaktische Fehler. Die Reproduktionen eingeprägter Programme enthielten verschiedene syntaktische Fehler, die eine Reihe von Design-Entscheidungen für die Darstellungsform von ABSYNT (vgl. die Abbildung) beeinflußten. In den Reproduktionen befanden sich z.B. Parameterknoten (Blätter) in der Position von Operatoren (Nicht-Blätter) und umgekehrt; ein Fehler, der eine graphische Unterscheidung von Parametern bzw. Konstanten einerseits und Operatoren andererseits nahelegte. Wie die Abbildung zeigt, sind die Eingabefelder von Blättern (Parameter, Konstanten) durch schwarze Balken gesperrt.

Semantische Fehler. Etwa 20 % der Berechnungen waren fehlerhaft. Mit Hilfe der dokumentierten Entwicklungssequenz des Regelsystems, das die Semantik von ABSYNT spezifiziert, versuchten wir nachträglich, die Berechnungsfehler zu systematisieren. In dieser Sequenz war die Menge berechenbarer Programme nach und nach erweitert worden: a) durch Hinzunahme neuer Regeln (z.B. die Erweiterung der Menge primitiver Operatoren), b) durch Ausdifferenzierung von Regeln oder Regelteilen (z.B. die Einführung von Funktionsrahmen), sowie c) durch Erweiterung des Anwendungsbereichs von Regeln (z.B. die Ersetzung spezifischer durch allgemeine Konzepte, wie Blatt und Wurzel). Hiermit korrespondierend ließen sich die meisten der gefundenen Berechnungsfehler beschreiben als

- fehlende Regeln: z.B. ist ein Operator unbekannt.

- Übergeneralisierung von Regeln: z.B. wird, wie bei der Rekursion erforderlich, nicht zwischen verschiedenen Inkarnationen eines Funktionsrahmens unterschieden. In diesem Fall wird die Berechnung mit dem Erfülltsein des Abbruchkriteriums als beendet betrachtet; eine Nachberechnung findet nicht statt.

- Überdifferenzierung des Anwendungsbereichs von Regeln: z.B. wird die Regel zur Behandlung von Aufrufen zwar bei nichtrekursiven, nicht aber bei rekursiven Aufrufen angewendet. In dieser Situation wird ein Reparaturversuch unternommen ("tinkering"; Brown & van Lehn, 1980): Der rekursive Aufruf wird z.B. als primitiver Operator behandelt, oder die Berechnung wird an der Stelle des rekursiven Aufrufs abgebrochen und in dem

nicht zu berechnenden Zweig der Fallunterscheidung fortgesetzt.

Erwerb, Differenzierung und Erweiterung des Anwendungsbereichs von Regeln als hypothetische Lernmechanismen (vgl. Norman, 1978; Goldstein, 1982) stellen in Verbindung mit der Repair-Theorie einen möglichen Ansatz zur Beschreibung semantischer Fehler dar.

2.2 Instruktionen zur Semantik von ABSYNT

Die Untersuchung hat, die Semantik von der Formularsprache betreffend, zwei Aspekte deutlich gemacht:

- Die Programmiersprache ist trotz ihrer zum Zeitpunkt der Untersuchung nur verbal beschriebenen Syntax und Semantik in angemessener Zeit erlernbar. Andererseits treten semantische Fehler auf, die ohne empirische Voruntersuchungen kaum antizipierbar sind, und die die Notwendigkeit des Ansetzens der Wissensdiagnose auch auf der semantischen Ebene unterstreichen. Geschieht dies nicht, so werden Programmierfehler schwer analysierbar, da sie sowohl auf Planungs- als auch auf semantische Fehler rückführbar sein könnten.

- Es ist unklar geblieben, ob und inwiefern semantische Fehler durch Fehler oder Ungenauigkeiten in den (verbal formulierten) Berechnungsinstruktionen zustande gekommen sind.

Die Vermittlung fundierten semantischen Wissens erscheint uns daher wesentlich (vgl. du Boulay & O'Shea, 1981; Neber, 1985). Aus diesem Grunde wurden drei verschiedene Regelsysteme für die Semantik von ABSYNT und dazugehörige graphische Repräsentationen entwickelt. Aufgrund psychologischer Überlegungen (vgl. auch Larkin & Simon, 1987) wurde eines dieser Regelsysteme und die dazugehörige graphische Repräsentation für die Entwicklung von Instruktionsmaterial ausgeschlossen. Für die beiden anderen Regelsysteme wurden Berechnungsinstruktionen entwickelt (Möbus & Thole, im Druck; Möbus & Schröder, im Druck). Dabei spielt das Merkmal von ABSYNT, Berechnungsabläufe vollständig sichtbar machen zu können, eine entscheidende Rolle: Die Instruktionen sind graphische Übersetzungen

der Regelsysteme und beschreiben jeden auf dem Bildschirm sichtbaren Berechnungsschritt (z.B. das Ersetzen eines Fragezeichens durch einen Wert).

3. Weitere Planungen

Mit den beiden Berechnungsinstruktionen sind vergleichende Untersuchungen zu ihrer Effektivität und zum Prozeß des Erwerbs von Berechnungswissen geplant. Dabei soll der Lernende explorativ vorgehen und die Instruktionen selbstgesteuert nutzen können. Nur bei fehlgeschlagenen Reparaturversuchen (Brown & van Lehn, 1980) soll von außen eingegriffen werden.

Erst wenn der Lernende ABSYNT-Programme richtig berechnen kann, werden ihm Programmieraufgaben vorgelegt. Mit diesem Vorgehen wird eine diagnostische Abgrenzung von semantischem und Planungswissen im Sinne einer Einengung diagnostischer Hypothesen angestrebt. Voruntersuchungen zum Programmieren mit nicht optimierten Instruktionen ergaben sehr heterogene und schwer beschreibgare Vorgehensweisen. Sie legten nahe, daß das Planen ebenfalls mit Instruktionen unterstützt werden sollte (Dosch, 1984; Soloway, 1986).

Literatur

Adelson, B., Problem solving and the development of abstract categories in programming language. Memory and Cognition, 9, 1981, 422-433

Anderson, J.R., Farrell, R. & Sauers, R., Learning to program in LISP. Cognitive Science, 8, 1984, 87-129

Bauer, F.L. & Goos, G., Informatik, 1. Teil. Berlin: Springer, 1982 (3. Auflage)

Brooks, R., Toward a theory of cognitive processes in computer programming. Int. J. of Man-Machine Studies, 9, 1977, 737-751

Brooks, R., Toward a theory of comprehension of computer programs. Int. J. of Man-Machine Studies, 18, 1983, 543-554

Brown, J.S. & van Lehn, K., Repair theory: A generative theory of bugs in procedural skills. Cognitive Science, 4, 1980, 379-426

Dosch, W., New prospects in teaching programming language. In F.B. Lovis & E.D. Tagg (Eds.), Informatics education for all students at University level. Elsevier Sciences Publ. B.V. (North Holland), IFIP, 1984, S. 153-169

du Boulay, B. & O'Shea, T., Teaching novices programming. In M.J. Coombs & J.L. Alty (Eds.), Computing skills and the user interface. New York: Academic Press, 1981, S. 147-200

Goldstein, I.P., The genetic graph: A representation for the evaluation of procedural knowledge. In D. Sleeman & J.S. Brown (Eds.), Intelligent tutoring systems. New York: Academic Press, 1982, S. 51-77

Hoc, J.M., Role of mental representation in learning a programming language. Int. J. of Man-Machine Studies, 9, 1977, 87-105

Hollan, J.D., Hutchins, E.L., McCandless, T.P., Rosenstein, M. & Weitzman, L., Graphical interfaces for simulation. ICS-Report 8603, University of California, San Diego, May 1986

Hoppe, H.U., Anforderungen an Programmiersprachen für den Unterricht unter dem Gesichtspunkt des interaktiven Programmierens. In H. Mandl & P.M. Fischer (Hg.), Lernen im Dialog mit dem Computer. München: Urban & Schwarzenberg, 1985, S. 191-209

Larkin, J.H. & Simon, H.A., Why a diagram is (sometimes) worth ten thousand words. Cognitive Science, 11, 1987, 65-99

Letovsky, S., Cognitive processes in program comprehension. In E. Soloway & S. Yvengar (Eds.), Empirical studies of programmers. New York: Ablex, 1986, S. 58-80

Möbus, C., Die Entwicklung zum Programmierexperten durch das Problemlösen mit Automaten. In H. Mandl & P.M. Fischer (Hg.), Lernen im Dialog mit dem Computer. München: Urban & Schwarzenberg, 1985, S. 140-154

Möbus, C. & Schröder, O., Knowledge specification and instructions for a visual computer language. Workshop on Knowledge Representation and Information Processing in Honor of the Sixtieth Birthday of Friedhart Klix. Zeitschrift für Psychologie (im Druck)

Möbus, C. & Thole, H.J., Instruktionsspezifikation bei einem Tutorsystem. In T. Christaller (Hg.), KIFS 87. Fünfte Frühjahrsschule Künstliche Intelligenz. Berlin: Springer (im Druck)

Neber, H., Psychologische Untersuchungen zum Lehren und Lernen des Computer-Programmierens. In H. Mandl & P.M. Fischer (Hg.), Lernen im Dialog mit dem Computer. München: Urban & Schwarzenberg, 1985, S. 218-228

Norman, D.A., Notes toward a theory of complex learning. In A.M. Lesgold, J.W. Pellegrino, S.D. Fokkema & R. Glaser (Eds.), Cognitive psychology and instruction. New York: Plenum Press, 1978, S. 39-48

Schmitt, H. & Wohlfarth, P., Mathematik-Buch 5 N. Mün-

chen: Bayerischer Schulbuch-Verlag, 1978

Shneiderman, B., Empirical studies of programmers: The territory, paths, and destinations. In E. Soloway & S. Yvengar (Eds.), Empirical Studies of Programmers. New York: Ablex, 1986, S. 1-12

Soloway, E., Learning to program = learning to construct mechanisms and explanations. Communications of the ACM, 29, 9, 1986, 850-858

Soloway, E., Ehrlich, K., Bonar, J. & Greenspan, J., What do novices know about programming? In A. Badre & B. Shneiderman (Eds.), Directions in human-computer interaction. Norwood: Ablex, 1984, S. 27-54

WORKSHOP

"Intelligente Lernsysteme"

vom 25. - 26.5.1987

in Tübingen

Abstracts

Herausgeber:

Rul Gunzenhäuser
Universität Stuttgart
Institut für Informatik

Heinz Mandl
Deutsches Institut für Fernstudien
an der Universität Tübingen