D R I T T E R W O R K S H O P

der Fachgruppe "Intelligente Lernsysteme"
der Gesellschaft für Informatik e.V.

vom 8. - 9.6.1989

in Tübingen

A b s t r a c t s

H e r a u s g e b e r :

Rul Gunzenhäuser
Universität Stuttgart
Institut für Informatik

Heinz Mandl
Deutsches Institut für Fernstudien
an der Universität Tübingen

Konzeptualisierung eines problemlösezentrierten Hilfesystems

Gabriele Janke, Claus Möbus und Heinz-Jürgen Thole

Projekt ABSYNT*, Fachbereich Informatik
Universität Oldenburg

Übersicht

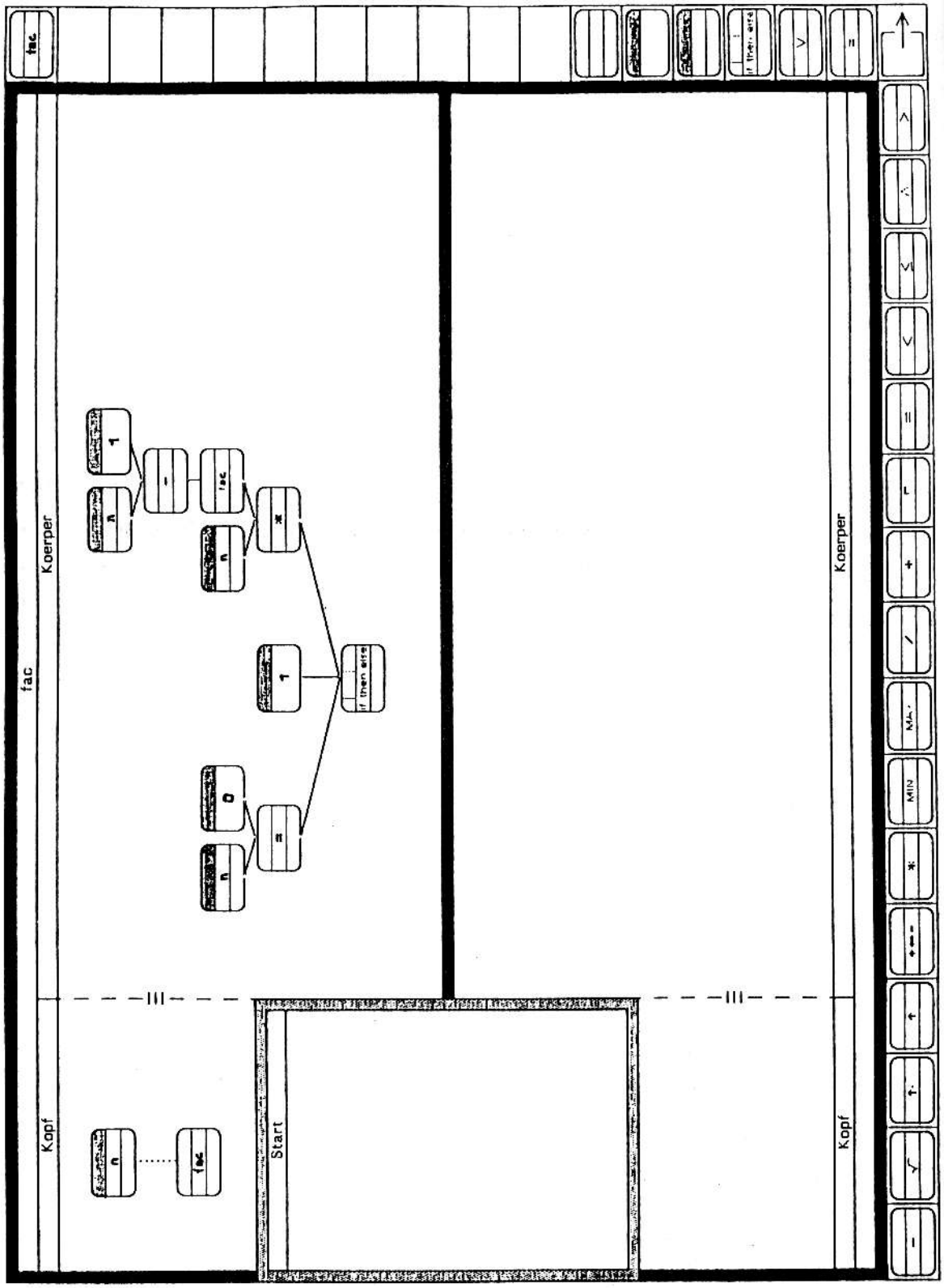
Die hier vorgestellte Untersuchung liefert einen Beitrag zum Thema Tutor- und Hilfesysteme beim Erwerb von Programmierwissen. Anhand einer Untersuchung, in der Versuchspersonen Programmieraufgaben in der grafischen rein funktionalen Programmiersprache ABSYNT lösten, wurden Anforderungen an ein individualisiertes Hilfesystem formuliert und anschließend eine erste Version für das Hilfesystem erarbeitet. Dieses basiert auf einem zeitgleich zu der Untersuchnung geschriebenen Diagnostikprogramm, das Fehler erkennen und fehlende Teile von ABSYNT-Programmen ergänzen kann.

Die Versuchspersonen planten und lösten die Programmieraufgaben in einer Aufgabensequenz, die gleichzeitig zum Erwerb der nötigen Programmierkenntnisse diente.

Eingliederung in das Projekt

Das grundlagenorientierte Ziel unseres Projekts ist die Erforschung kognitiver Prozesse beim Erwerb von Programmierwissen, speziell die Analyse von Problemlöseprozessen beim Programmieren. Unter Problemlösen

* geördert durch die Deutsche Forschungsgemeinschaft, Schwerpunktprogramm Wissenspsychologie, Förderungsnummer Mo 292/3


fac
Koerper
Kopf
Start
Koerper
Kopf
fac
n
1
-
fac
n
*
1
if then else
0
n
=
n
fac
MIN
MAX

verstehen wir das Planen von Programmen, die Fehlersuche und -korrektur.

Das anwendungsbezogenen Ziel unseres Projekts - Thema dieses Beitrags - ist die Entwickling eines adaptiven on-line Hilfesystems, bestehend aus einer Diagnosekomponente, die die Problemsituationen analysiert und einer Hilfekomponente, die individuelle Hilfen anbietet.

Die Problemlöseumgebung "ABSYNT"

ABSYNT bietet eine visuelle, rein funktionale Programmiersprache. Sie entstand aus Ideen von Bauer und Goos (1982), die fertige funktionale Programme in Form von Baum-Diagrammen anschaulich machten. In unserem Projekt wurde aus diesen illustrierenden Diagrammen eine lauffähige Programmiersprache mit direkt manipulierbaren Objekten entwickelt (Janke,Kohnert 1989).

Abb.1 zeigt eine Momentaufnahme beim Programmieren mit ABSYNT. ABSYNT-Programme sind Rahmen. Ein Rahmen entspricht einer Funktion in einer funktionalen Programmiersprache.

Jeder Rahmen gliedert sich in einen Kopf und einen Körper. Im Kopf wird der Name des Rahmens, der ihn repräsentierende Operatorknoten (selbstdefinierter höherer Operatorknoten) und die Anzahl und Reihenfolge der Parameter des Rahmens festgelegt. Im Körper wird die Rechenvorschrift definiert. Dies geschieht in Form von ABSYNT-Bäumen. Dies sind Bäume, die aus aus Operatorknoten (repräsentieren Operatoren), Konstanten-Knoten (repräsentieren Konstanten), Parameterknoten (repräsentieren Parameter) und Verbindungslinien zwischen den Knoten aufgebaut sind.

Das Startfenster von ABSYNT entspricht dem Top Level von Lisp. Es kann ebenfalls ABSYNT-Bäume, allerdings ohne Parameterknoten, enthalten. Damit können Programmaufrufe repräsentiert werden.

Der Programmierer baut die Bäume in den Rahmen und im Start mit den in der Knotenleiste zur Verfügung gestellten Knoten auf. Diese werden von ihm verbunden und - falls nötig - beschriftet. Veränderungen können auch durch Verschieben und Löschen der Knoten und Linien herbeigeführt werden. Abb.1 zeigt einen Zustand des Editors, in dem ein korrektes Programm (Fakultät, rekursiv) im oberen Rahmen erstellt wurde.

Die Berechnung des Programms durch den Interpreter kann schrittweise mit einem Trace demonstriert werden. Mit Hilfe dieses Trace gelang es den Versuchspersonen, alle syntaktischen und viele semantische Fehler zu lokalisieren.

ABSYNT ist in Interlisp/Loops auf einer Siemens 5822 implementiert.

Die Aufgabensequenz

Die Unterrichtsreihe, die gleichzeitig dem Erwerb von Programmierwissen für ABSYNT und der Bearbeitung von Problemen diente, wurde nach Ideen von VanLehn (1987) und Zhu und Simon (1987) entwickelt: Das Wissen, das in den aufeinanderfolgenden Teilen der Unterrichtsreihe erworben wird, baut im Sinne von Gagné (1973) aufeinander auf (Janke 1989/1). Die Untersuchungsergebnisse zeigen, daß bis auf weiteres bei der Analyse von Problemsituationen von der Arbeitshypothese ausgegangen werden kann, daß Fehler und Probleme durch die gerade neu zu erwerbenden Lernziele verursacht werden.

Die Unterrichtsreihe wurde so konzipiert, daß die Versuchsleiter so wenig wie möglich in den Wissenserwerbs- und Problemlöseprozeß eingreifen mußten. Dies garantierte einheitliche Bedingungen für alle Versuchspersonen und eine Vielfalt individueller Lösungen.

Beispiele für Problemsituationen aus der Untersuchung

An drei Problemsituationen, die alle bei der Lösung der letzten und schwierigsten Aufgabe der Unterrichtsreihe

entstanden sind, sollen beispielhaft einige Typen von Problemsituationen dargestellt werden (Janke, 1989/2).

Die Aufgabe besteht darin, ein Programm zu schreiben, das für jede natürliche Zahl prüft, ob sie gerade ist (kurz "gerade" genannt).

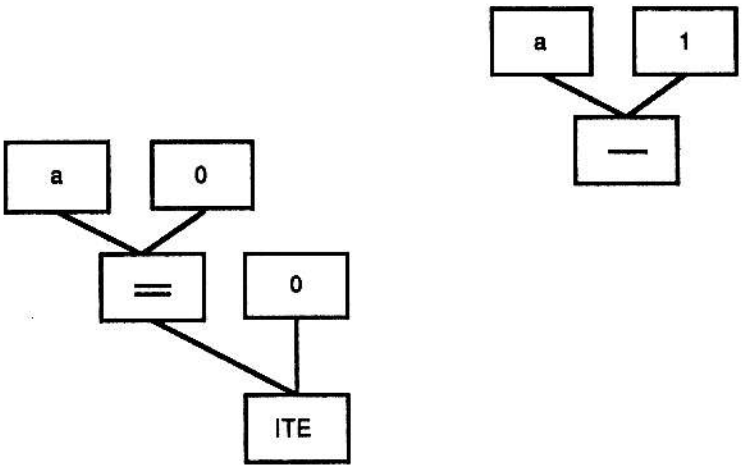


Abb.2

In der ersten Problemsituation, von der hier nur der Körper gezeigt wird (der Kopf des Rahmens war noch leer), lag noch ein syntaktisch unvollständiger Baum vor. Neben den Verbalisierungen deuten der Fallunterscheidungsoperator (ITE = If Then Else), die Stop-Bedingung im if-Zweig und die Reduktion des Paramters (unverbundener Teilbaum oben rechts) darauf hin, daß die Versuchsperson das Problem rekursiv lösen wollte. Der then-Zweig (Mitte) enthält einen Fehler: statt der Konstante "T" (True) wurde die Konstante "0" gewählt.

Aus den folgenden Gründen verfolgten wir allerdings die Strategie, bei Fehlern nicht sofort einzugrefen:

- wir wollten untersuchen, wie die Versuchspersonen Problemsituationen bewältigen (dazu gehört auch die Fehlersuche und - korrektur).

- die Schüler sollten Fähigkeiten zur Fehlersuche und - korrektur erwerben

- wie im vorliegenden Fall kam es häufig vor, daß Fehler im weiteren Verlauf der Programmentwicklung selbst bemerkt werden.

Das verbalisierte Problem der Situation lag zudem nicht in der fehlerhaften Stelle, sondern in der Programmierung des rekursiven Aufrufs und seiner Rekombination. D.h. die Versuchsperson suchte geeignete Operatoren zur Vervollständigung des Programms.

Sie löste das Problem zunächst, indem sie die Teillösung für den rekursiven Aufruf von der vorigen Aufgabe (Rekursion mit zwei Paramtern) übernahm.

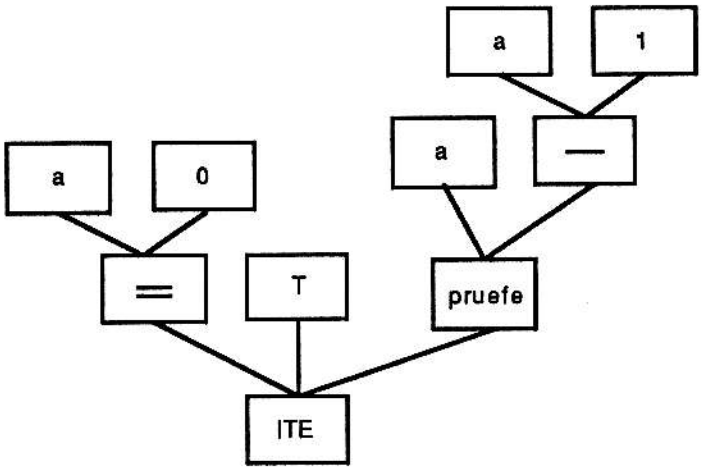


Abb.3

Es lag nun ein syntaktisch vollständiger Baum vor. Der Fehler der ersten Problemsituation wurde korrigiert, aber der Körper enthielt jetzt einen anderen Fehler: der rekursive Aufruf hat zwei Eingänge, was zu keiner naheliegenden Lösung führen kann. Dieser Fehler ist eindeutig auf die schon mehrfach benutzte Strategie der Versuchsperson zurückzuführen, analoge Lösungen zu übergeneralisieren. Die Strategie wurde verwendet, weil das Lernziel "Programmieren eines rekursiven Aufrufs mit der richtigen Anzahl von Eingängen" noch nicht beherrscht wird.

Der fehlerhafte Programmentwurf einer anderen Versuchsperson zeigt einen anderen Typ von Problemsituationen:

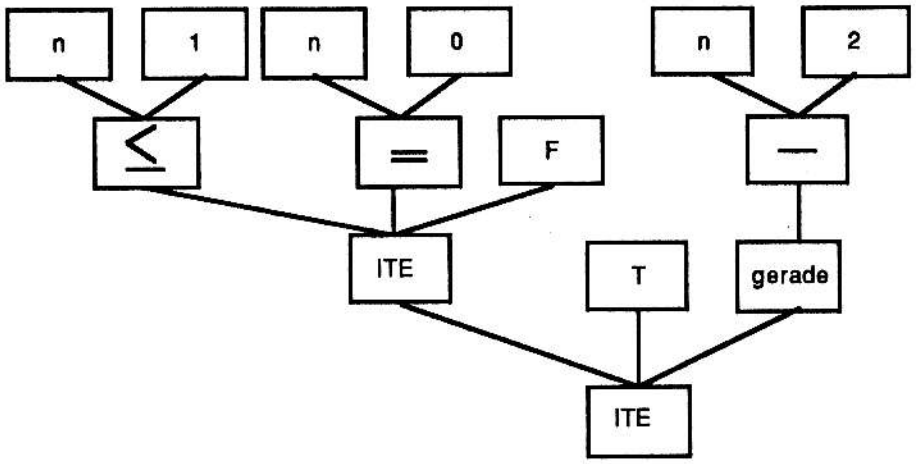


Abb.4

Dieser Entwurf enthält alle Teilbäume, die zu einer richtigen Lösung benötigt werden. Das Problem lag offensichtlich in der Kombination des Stops der Rekursion und dem rekursiven Aufruf, was zu der falschen Verwendung einer zweiten Fallunterscheidung (ITE) führte. In diesem Fall war der Fehler wahrscheinlich darauf zurückzuführen, daß die Versuchsperson die Kompakheit der Stopbedingung und ihres Wertes nicht erkannt hat.

Anforderungen an das Hilfesystem

Aus der Analyse aller in der Untersuchung vorgekommenen Problemsituationen (Janke 1989/2) ergaben sich folgende Anforderungen an ein individualisiertes Hilfesystem, die in den Punkten 4 und 5 mit den dort genannten Autoren im Einklang stehen:

1. Auf eine große Vielfalt von verschiedenen Lösungen muß eingegangen werden können.

2. Alle Typen von Problemsituationen müssen behandelt werden können: 1)syntaktisch unvollständige a)schon fehlerhafte, b)bisher korrekte, 2)syntaktisch

vollständige fehlerhafte und 3)korrekte aber umständlich Lösungsansätze.

3. In allen Problemsituationen müssen die Programme bzw. Programmteile diagnostiziert werden können, d.h. von Teilbäumen muß auf angestrebte Teillösungen (Teilziele) inferiert werden können.

4. In allen Problemsituationen müssen prinzipiell Hilfen generiert werden können. Für die Entscheidung über Zeitpunkt und Art der Hilfestellung müssen Schülerdaten berücksichtigt werden, z.B.: aktueller Problembereich, schon erreichte Lernziele, frühere Fehler, frühere Strategien,... (Anderson,1989, Clancey,1986, Corbett,1989, Moll,1989).

5. Die Hilfen sollten in abgestufter Form gegeben werden können (vom allgemeinen Hinweis zur konkreten Information) und diese Abstufung sollte indiviuell einstellbar sein (Moll, 1989).

Vorversion des Hilfesystems

Das Hilfesystem wird auf dem parallel zu der Untersuchung entwickelten Diagnostikprogramm basieren. Dieses bildet eine Ziel-Mittel-Relation, die das Parsen und Generieren von Lösungen realisiert. Die Ziel-Mittel-Relation kann auch als UND/ODER-Graph im Sinne von Nilsson (1980) interpretiert werden (Möbus und Schröder, 1989).

Abb.5 zeigt einen Ausschnitt des Und/Oder-Graphen, der das Prinzip des Diagnostikprogramms verdeutlicht (im folgenden UOG genannt). Dieser Ausschnitt befaßt sich mit dem Körper des Rahmens, der einige Lösungen für die "gerade"-Aufgabe generieren und parsen kann.

Zunächst die Erklärung der Symbole:

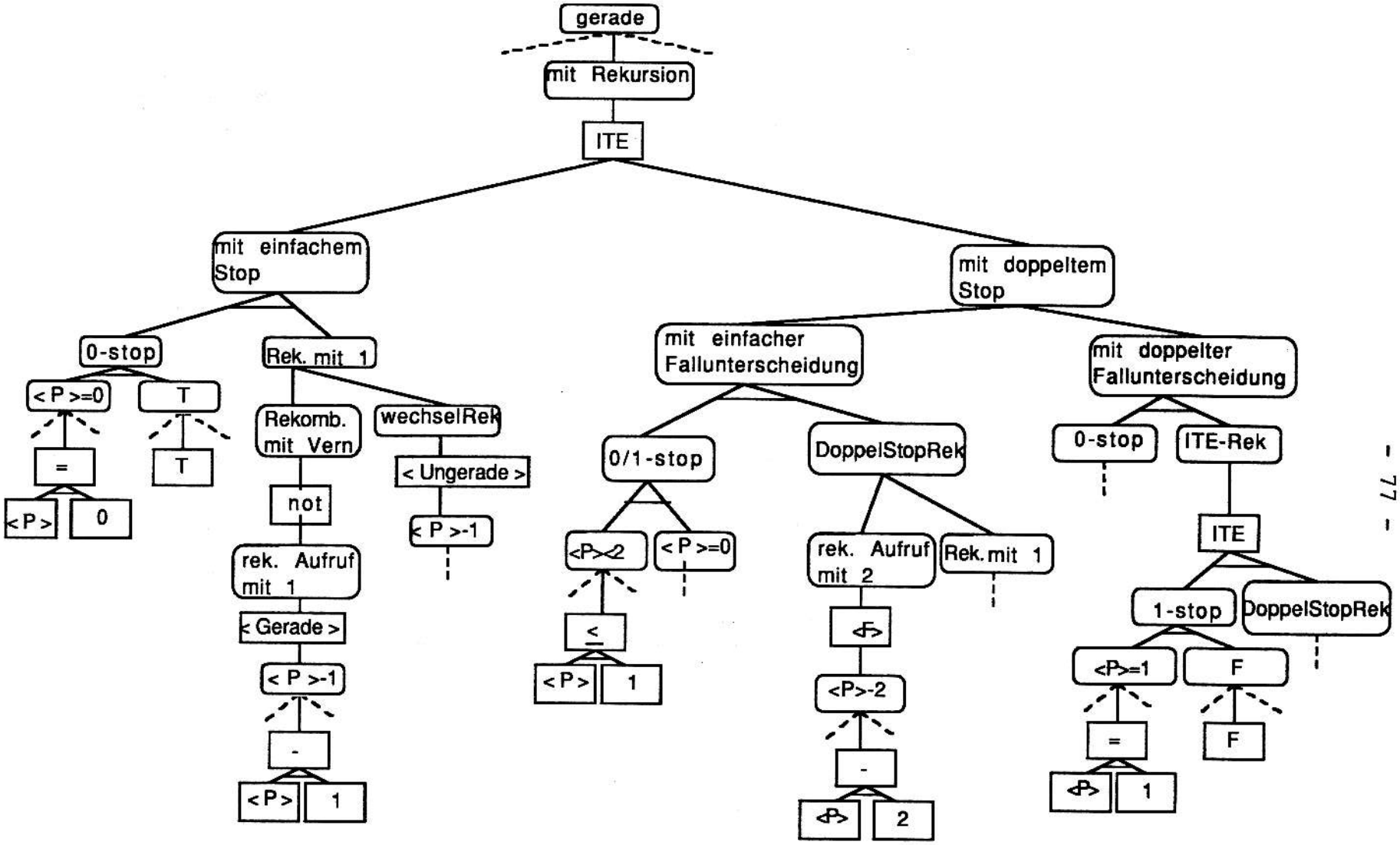

gerade
mit Rekursion
ITE
mit einfachem Stop
mit doppeltem Stop
0-stop
Rek. mit 1
< P >=0
T
=
< P >
0
T
Rekomb. mit Vern
wechselRek
not
< Ungerade >
rek. Aufruf mit 1
< P >-1
< Gerade >
< P >-1
-
< P >
1
mit einfacher Fallunterscheidung
mit doppelter Fallunterscheidung
0/1-stop
DoppelStopRek
<P><2
< P >=0
<
=
< P >
1
rek. Aufruf mit 2
Rek. mit 1
<P>
<P>-2
-
<P>
2
0-stop
ITE-Rek
ITE
1-stop
DoppelStopRek
<P>=1
F
=
<P>
1
F

Abb. 5

Die Rechtecke repräsentieren Ziele, die als Knoten im ABSYNT-Baum dargestellt sind (terminale Symbole im Sinne einer Grammatik).

Die abgrundeten Figuren repräsentieren Ziele, die noch nicht direkt als Knoten des ABSYNT-Baums dargestellt sind (nicht terminale Symbole).

Verzweigungen, die durch einen Bogen verbunden sind, sind Und-Verzweigungen.

Verzweigungen, die nicht durch einen Bogen verbunden sind, sind Oder-Verzweigungen.

Eine konkrete Lösung, d.h. ein auf dem Kopf stehender ABSYNT-Baum, wird generiert, indem von oben kommend entlang jeweils einer Oder Verbindung und aller Und-Verbindungen die konkreten ABSYNT-Knoten verbunden werden. Endet dabei ein Zweig des Zielbaums bei einem Teilziel in einer runden Figur, dann muß man zu der Stelle im Baum springen, wo dieses Teilziel weiter differenziert wird.

Umgekehrt können ABSYNT-(Teil-)Bäume (Teil-)Zielen des Zielbaums zugeordnet werden.

In der in Abb.2 gezeigten Problemsituation kann z.B. der if-Zweig der Fallunterscheidung dem Teilziel "0-stop" des Zielbaums zugeordnet werden, für den bisher fehlerhaften then-Zweig kann die richtige Ergänzung geliefert werden. Der rechte Teilbaum "a - 1" kann dem Teilziel "<P> - 1" zugeordnet werden. Dieses Teilziel kann dem Teilziel "Rek. mit 1" untergeordnet werden. Auf diese Weise sind zwei Gruppen von Ergänzungen generierbar ("Rekomb. mit Vern" und "wechselRek"). Insgesamt kann durch Parsen und Generieren der fehlerhafte und unvollständige Programmentwurf zu einer Vielzahl richtiger Lösung ergänzt werden. Bei der Hilfestellung sollte daraus unter Berücksichtigung eines Schülermodells der für den Schüler optimale Vorschlag ausgewählt werden.

Da die automatische Präsentation einer korrekten Ergänzung mit dem Charakter eines Problemlösemonitors nicht vereinbar ist, soll mit Hilfe der im UOG enthaltenen Teilziele ein abgestuftes Hilfesystem entwickelt werden, das fehlende Programmteile mit Teilzielen in Beziehung setzt (Moll, 1989).

Weitere Forschung

Bei der Generierung von Hilfen entstehen zwei Schwierigkeiten:

1. die Auswahl der für den Schüler geeigneten generierten Ergänzung aus der Menge aller möglichen Ergänzungen

2. die Verbalisierung von Hinweisen, die nicht einfach die Lösung zeigen, sondern die als Erklärung akzeptiert werden

Für die Auswahl der optimalen Schülerhilfe müssen Schülerdaten (Clancey, 1986) einbezogen werden, die als Indikatoren für seinen Wissensstand gelten können.

Hinweise für mögliche Ergänzungen wollen wir aus den Teilzielen des UOG ableiten. Dazu ist es notwendig, die bisher nur aus der Aufgabenanalyse entstandene Zielstruktur empirisch zu validieren, so daß man von einer psychologisch fundierten Intentionsdiagnostik im Sinne von Johnson (1986) sprechen kann.

Literatur

Anderson, J.R., "Psychology and Intelligent Tutoring", in: Bierman, Breuker, Sandberg (Hrsg.) Proceedings of the 4th International Conference on AI in Education, in Amsterdam, 24-26 März 1989, Amsterdam: IOS, 1989, 1

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

Clancey, W.J. "Qualitative Student Models", Annual Review of Computer Science, 1986, Vol.I, 381 - 450

Corbett, CA.T. und Anderson, J.R, "Feedback Timing and Student Control in the Lisp Intelligent Tutoring System", in: Bierman, Breuker, Sandberg (Hrsg.) Proceedings of the

4th International Conference on AI in Education in Amsterdam, 24-26 März 1989, Amsterdam: IOS, 1989, 64 - 72

Gagné, R.M., "Der Erwerb von Wissen" in: Hofer, Weinert (Hrsg.) Pädagogische Psychologie 2, Fischer Taschenbuchverlag, Frankfurt a.M., 1973, 106- 123

Janke,G. "Voruntersuchung zum Programmieren mit ABSYNT, Teil 1: Aufbau der im Versuch verwendeten Aufgabensequenz", ABSYNT Memo 1/89, Universität Oldenburg, 1989

Janke,G. "Voruntersuchung zum Programmieren mit ABSYNT, Teil 2: Versuchsbeschreibung und Analyse aller Problemsituationen", ABSYNT Memo 2/89, Universität Oldenburg, 1989

Janke, G. und Kohnert, K."Interface design of a visual programming language: evaluating runnable specifications", in: Klix, Streitz, Waren, Wandtke (Hrsg.), MACINTER II Man-Computer Interaction Research, Proceedings of the Second Network Seminar of MACINTER, Berlin/DDR, 21. - 25. März 1988, Amsterdam: North Holland, 1989, 567 - 581

Johnson, W.L., "Intention-Based Diagnosis of Novice Programming Errors", Los Altos, CA.: Morgan Kaufmann Publishers, 1986

Möbus, C. und Schröder, O. "Knowledge specification and instruction for a visual computer language", in: Klix, Streitz, Waren, Wandtke (Hrsg.), MACINTER II Man-Computer Interaction Research, Proceedings of the Second Network Seminar of MACINTER, Berlin/DDR, 21.-25.März 1988, Amsterdam: North Holland, 1989, 535 - 565

Möbus, C. und Schröder, O. "Zur Entwicklung des Problemlösemonitors ABSYNT: Wissenserwerbsmodellierung und Hilfekomponente", in: Kreowski, Krieg-Brückner (Hrsg.) Berichte aus dem Fachbereich Informatik der Universtät Bremen, 2. Tagung zur Küsteninformatik, 18.-20. Mai 1989 in Bederkesa (im Druck)

Moll, Th."Über die Verbesserung der Benutzerunterstützung durch ein Online-Tutorial", in Maaß, Oberquelle (Hrsg.) Software-Ergonomie ´89, Stuttgart:Teubner 1989, 223 - 232

Nilsson, N. "Principles of Artificial Intelligence", Palo Alto, Ca.: Tioga Press, 1980

VanLehn, K.:"Learning one Subprocedure per Lesson" Artificial Intelligence 31 (1987) 1 - 40

Zhu,X. und Simon,H. A.:"Learning mathematics from examples and by doing", interner Forschungsbericht der Chinese Academy of Sience und der Carnegie-Mellon University, 5.März 1987