Regel für while-Schleifen:
Um

{E} P; while B do Q; R {A}

zu zeigen, genügt es, eine Aussage I ("Schleifeninvariante"), eine noethersche Relation $\prec$ auf einer Menge M und einen Term t ("Terminationsterm") zu suchen, für die man folgendes beweisen kann:

{E} P {I},
aus (I und B) folgt $t \in M$,
{I und B und t=T} Q {I und $t \prec T$} (T .. eine neue Variable),
{I und nicht B} R {A}.

Beispiel für eine Programmverifikation: Betrachte den Euklidischen Algorithmus zur Bestimmung des größten gemeinsamen Teilers (GGT) von zwei natürlichen Zahlen m, n zusammen mit seiner Spezifikation:

{m,n $\in$ **N**}
(z,r):= (m,n)
(*)
while r≠0 do (z,r):= (r,Rest(z,r))
{z = GGT(m,n)}.

Hier steht "Rest(z,r)" für den Rest bei der ganzzahligen Division von z durch r. Als induktive Behauptung an der Stelle (*) wählen wir

GGT(z,r) = GGT(m,n), z≠0 oder r≠0,

als Menge M mit noetherscher Relation wählen wir **N** mit der Kleinerbeziehung und als Terminationsterm r. Gemäß der Regel für while-Schleifen muß man dann zeigen:

(1) { m,n$\in$**N** } (z,r):= (m,n) { GGT(z,r)=GGT(m,n), z≠0 oder r≠0 },
(2) aus GGT(z,r)=GGT(m,n), z≠0 oder r≠0, r≠0 folgt r$\in$**N**,

(3) { GGT(z,r)=GGT(m,n), z≠0 oder r≠0, r≠0, r=T }
(z,r):= (r,Rest(z,r))
{ GGT(z,r)=GGT(m,n), z≠0 oder r≠0, r≤T },
(4) { GGT(z,r)=GGT(m,n), z≠0 oder r≠0, r=0 } { z=GGT(m,n) }.

(1), (2) und (4) sind leicht zu zeigen. (3) wird durch Anwenden der "Zuweisungsregel" aufgelöst in

(3') Aus GGT(z,r)=GGT(m,n), z≠0 oder r≠0, r≠0, r=T
folgt GGT(r,Rest(z,r))=GGT(m,n), r≠0 oder Rest(z,r)≠0,
Rest(z,r)≤T.

(3') folgt aber unmittelbar aus dem folgenden Wissen über den GGT:

(W) aus r≠0 folgt GGT(z,r)=GGT(r,Rest(z,r)).

Wir beobachten an dem Beispiel einige für die automatische Programmverifikation grundsätzlich wichtige Dinge:

1. Die Erstellung der zu beweisenden Aussagen der Art (3') etc. ("Verifikationsbedingungen") aus der Problemspezifikation, dem Programm und den induktiven Behauptungen ist ein vollkommen mechanischer Vorgang, der leicht automatisiert werden kann.

2. Der Beweis der Verifikationsbedingungen zerfällt in einen wesentlichen Teil, der im wesentlichen das algorithmisch brauchbare Wissen benutzt, das auch zum Entwurf des Algorithmus (z.B. in Form eines logischen Programms) zentral ist (siehe (W) im Beispiel), und in triviale Teile, wie z.B. der Beweis von (1), der im wesentlichen mit leichten Substitutionen etc. auskommt.

3. Der "triviale" Teil ist leicht automatisierbar und es ist auch hilfreich, diese Routineteile des Beweises zu automatisieren. Der "wesentliche Teil" kann auch leicht automatisiert werden, sobald das nötige algorithmische Wissen zur Verfügung gestellt wird. Der Beweis dieses Wissens kann allerdings beliebig

schwierig sein, in ihm steckt der "kreative" Teil des Algorithmenentwurfs.

4. Programmverifikation ist nur sinnvoll, wenn sie in den Entwurfsprozeß eingebettet ist. Das algorithmische Wissen, das den Entwurfsprozeß leitet, ist eben auch dasjenige, welches über die zentralen Stellen des Korrektheitsbeweises führt. Verifikation von "fertigen" Programmen bezüglich vorgegebener Spezifikationen ist "unnatürlich", weil man beim Verifizieren das Programm noch einmal entwickeln muß.

Literaturhinweise zur automatischen Programmverifikation: Das bisher am weitesten entwickelte Verifikationssystem ist der Stanford-PASCAL-Verifier, siehe (Luckham et al., 1979), (Polak 1981). Dieses System baut auf der Methode der induktiven Behauptungen auf und kann den Entwicklungsprozeß von logisch durchaus anspruchsvollen Algorithmen zu einem sehr großen Teil automatisch unterstützen. Andere fortgeschrittene Programmverifikationssysteme sind z.B.: das System von (Boyer, Moore 1979), das als Programmiersprache und als Beweissprache eine LISP-ähnliche Sprache verwendet; das AFFIRM-System, siehe (Gerhart et al., 1980), das auf dem Konzept der abstrakten Datentypen aufbaut (siehe die Bibliographie über abstrakte Datentypen (Kutzler, Lichtenberger 1983)); das LCF-System, das auf dem Lambda-Kalkül basiert, siehe (Gordon, Milner, Wadsworth 1979). Für eine detailierte Einführung in die Praxis der "händischen" Programmverifikation in Verbindung mit einer praktischen Einführung in die Technik des Beweisens siehe (Buchberger, Lichtenberger 1980). Leider gibt es noch sehr wenig zusammenfassende Literatur zu speziellen automatischen Beweisern, die im Rahmen der Programmverifikation sicher eine mindestens ebenso wichtige Rolle wie universelle Beweiser spielen. Ein Beispiel eines speziellen Beweiser ist (Nelson, Oppen 1980).

8.6 Ein integrierter Software-Arbeitsplatz

Die verschiedenen Ansätze zum automatischen Programmieren sind derzeit in verschiedenen experimentellen Pilotsystemen implementiert. In Zusammenhang mit den vielfältigen Bemühungen, das Management des Software-Entwicklungsprozesses durch den Computer
zu unterstützen, die Computer-Ein-/Ausgabe durch graphische und natürlichsprachliche Schnittstellen zu erleichtern, Expertenwissen in Expertensystemen und Methodenbanken zur Verfügung zu stellen, die Potenz zur Erarbeitung von algorithmisch brauchbarem Wissen durch universelle und spezielle automatische Beweiser zu vergrößern, gewisse Transformationsprozesse auf mathematischen Objekten zu automatisieren ("Computer-Algebra"), sollten in naher Zukunft modular aufgebaute Systeme entstehen, die den Problemlöseprozeß von der vagen Problemformulierung über die exakte Problemspezifikation im Rahmen einer formalen, deskriptiven Sprache bis hin zum korrekten, effizienten und auf einer abstrakten Maschine lauffähigen Programm unterstützen. Dabei sollte der Mensch weder bezüglich seines Problemlösestils eingeschränkt werden, sondern unter verschiedenen "Philosophien" angepaßt an das Problem und den Stand der Problemlösung wählen können, noch erscheint es sinnvoll, ihm alle kreativen Aktionen während des Problemlöseprozesses abzunehmen. Flexible Systeme, die die verschiedenen Ansätze und das damit gewonnene Know-How integrieren, erscheinen für die nächste Zukunft realisierbar und erstrebenswert. In (Buchberger 82) wird ein Studienschwerpunkt im Rahmen der Informatik bzw. Mathematik beschrieben, der versucht, die Studenten in die formalen und praktischen Aspekte solcher intergrierter Systeme einzuführen.

Dank: Diese Arbeit wurde durch den Österreichischen Fonds zur Förderung der wissenschaftlichen Forschung unterstützt (Projekt Nr. 4567).

Literatur

- Barr A., Feigenbaum E. A.: The Handbook of Artificial Intelligence. Vol. II, Heuristech Press, Stanford; 1982.

- Bauer F. L. und CIP Language Group: The Munich Project CIP, Vol.I: The Wide Spectrum Language 85. Bericht, Technische Universität München, Institut für Informatik; Dezember 1983.

- Bibel W.: Syntax-Directed, Semantics-Supported Program Synthesis. Artificial Intelligence 14, 243-261; 1980.

- Biermann A. W., Guiho G. (Hsg.): Computer Program Synthesis Methodologies. Proc. of the NATO Advanced Study Institute, Bonas, September 1981, Reidel Publ. Comp., Dodrecht, Boston, London; 1983.

- Boyer R. S., Moore J. S.: A Computational Logic. Academic Press, New York - London; 1979.

- Buchberger B.: Ein algorithmisches Kriterium für die Lösbarkeit algebraischer Gleichungssysteme. Aequationes mathematicae 4(3), 374-383; 1970. (Publikation der Dissertation, Univ. Innsbruck, 1965).

- Buchberger B.: Studienschwerpunkt CAMP (Computer-Aided Mathematical Problem Solving) an der Universität Linz. Bericht Nr. CAMP 82-4.1, Institut für Mathematik, Universität Linz; 1982.

- Buchberger B.: Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory. In: Recent Trends in Multidimensional Systems Theory (N. K. Bose Hsg.), D. Reidel Publ. Comp., Dodrecht, Boston, London; erscheint 1984.

- Buchberger B., Collins G. E., Loos R.: Computer-Algebra (Symbolic and Algebraic Computation). Springer-Verlag, Wien - New York; 1982 (2. Auflage 1983).

- Buchberger B., Lichtenberger F.: Mathematik für Informatiker I (Die Methode der Mathematik). Springer-Verlag, Berlin - Heidelberg - New York; 1980 (2. Auflage 1981).

- Burstall R. M., Darlington J.: A Transformation System for Developing Recursive Programs. J. ACM 24(1), 1977.

- Clark K. L., McCabe F. G.: Micro-Prolog: Programming in Logic. Prentice-Hall, Engelwood Cliffs, N.J.; 1984.

- Clark K. L., Tärnlund S.-A. (Hsg.): Logic Programming. Academic Press, London; 1982.

- Clocksin W. F., Mellish C. S.: Programming in Prolog. Springer, Berlin - Heidelberg - New York; 1981.

- Darlington J.: The Synthesis of Implementations for Abstract Data Types, A Program Transformation Tactic. In (Biermann, Guiho 1983), 309-334.

- Darlington J., Burstall R. M.: A System which Automatically Improves Programs. Acta Informatica 6, 41-60; 1976.

- Gerhart S. L. and AFFIRM group: An Overview of AFFIRM: A Specification and Verification System. Proc. of the IFIP Congress 1980 (Lavington S. H. Hsg.), 343-347; 1980.

- Goad C. A.: Automatic Construction of Special Purpose Programs. Proc. 6th Conference on Automated Deduction, Springer Lecture Notes in Computer Science 138 (Loveland D. W. Hsg.), Berlin - Heidelberg - New York - Tokyo; 1982.

- Gordon M. J., Milner A. J., Wadsworth C. P.: Edinburgh LCF. Lecture Notes in Computer Science 78, Springer, Berlin; 1979.

- Greibach S. A.: Theory of Program Structures: Schemes, Semantics, Verification. Lecture Notes in Computer Science 36, Springer, Berlin - Heidelberg - New York; 1975.

- Hesse W.: Methoden und Werkzeuge für Software-Entwicklung: Ein Marsch durch die Technologie-Landschaft. Informatik-Spektrum 4(4), 229-245; 1981.

- Jouannaud J.-P., Kodratoff Y.: Program Synthesis from Examples of Behavior. In: (Biermann, Guiho 1983), 213 - 250.

- Knuth D. E., Bendix P. B.: Simple Word Problems in Universal Algebras. Proc. of the Conf. on Computational Problems in Abstract Algebra, Oxford 1967 (Leech J., Hsg.), 263-298, Pergamon Press, Oxford; 1970.

- Kowalski R.: Logic for Problem Solving. North-Holland, New York - Oxford; 1979.

- Kutzler B., Lichtenberger F.: Bibliography on Abstract Data Types. Informatik Fachberichte 68, Springer, Berlin - Heidelberg - New York - Tokyo; 1983.

- Lescanne P.: Computer Experiments With the REVE Term Rewriting System Generator. Proc. of the Principles of Programming Languages Conference; 1983.

- Luckham D. C. and PASCAL Verifier Group: Stanford PASCAL Verifier User Manual. Stanford, Computer Science Department, Report No. STAN-CS-79-731; 1979.

- Manna Z., Waldinger R.: A Deductive Approach to Program Synthesis. ACM TOPLAS 2/1, 92-121; 1980.

- Manna Z., Waldinger R.: Deductive Synthesis of the Unification Algorithm. In: (Biermann, Guiho 1983), 251-308.

- Nelson C. G., Oppen D. C.: Fast Decision Procedures Based on Congruence Closure. J. ACM 27(2), 356-364; 1980.

- Polak W.: Program Verification at Stanford: Past, Present, Future. Report, Stanford University, Computer Systems Laboratory; 1981.

- Zima H.: Compilerbau II (Synthese und Optimierung). Reihe Informatik 37, Bibliographisches Institut, Mannheim, Wien, Zürich; 1983.