South African
Computer
Journal
Number 21
August 1998

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 21
Augustus 1998

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarvereniging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

World-Wide Web: `http://www.cs.up.ac.za/sacj/`

Subscriptions

	Annual	Single copy
Southern Africa	R50.00	R25.00
Elsewhere	US$30.00	US$15.00

An additional US$15 per year is charged for airmail outside Southern Africa

to be sent to:

Computer Society of South Africa
Box 1714, Halfway House, 1685
Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

Cultivating the information systems discipline

Niek du Plooy, Sub-Editor: Information Systems

Whether by 'information system' we mean a simple bookkeeping system for a small business, or a monolithic integrated 'management information system' for a global corporation, all organisations currently need information systems in order to function effectively. The computer and business community at large have readily adopted and accepted the use of the term 'information systems', but perhaps without too much real thought being given to any more profound meaning of the term. Departments bearing that name (or something very similar) are commonly found in organisations. But can the same be said for the 'academic' use of the term, as in describing the information systems discipline? Has it been 'accepted' as a separate scientific discipline?

The term 'discipline' is often loosely applied to indicate the scientific 'field', that is, the organised 'body of knowledge' or 'domains of discourse' within which (mainly) academic activities concerning a specific topic or a number of related topics, are conducted. [3] point out that a scientific discipline has a certain paradigm associated with it, meaning that researchers in that discipline are familiar with the research topics, the research methods and the accepted ways to interpret the results in their chosen field. A discipline is further strengthened and consolidated by the educational process whereby a researcher becomes a practitioner in that discipline, initially through the pursuit of academic degrees and thereafter, through recognition amongst his/her peers. Formal study in a particular discipline results in the value sets and exemplars (the 'paradigm') of that discipline being adopted by the student, either consciously or unconsciously.

Is 'information systems' truly a recognised scientific discipline such such as this? In the past, prominent authors such as Peter Keen did not think so [15, 16]. He deplored the lack of a cumulative tradition and advocated that one be built up, asking for a clarification of the reference disciplines of this new science and a definition of its dependent variable and the building of a cumulative tradition, amongst other things. [1] however, disputed Keen's position and pointed to strong links between research and practice found in their analysis. [11] showed clearly that 'orthodoxy' exists in many aspects of information systems, i.e. in information systems methodologies as well as in other areas of information systems development. This claim was supported by [13] who, in a detailed study based on papers in scientific journals, scientific conferences and textbooks, identified seven different but complementary 'schools of thought' within the field of information systems. In a study of leading universities and leading researchers in decision support systems, [9] provide exemplars, at least for that particular sub-discipline. [5] conclude from a citation study of journal influence during the period 1981 through 1985, that the discipline of information systems has attained stability and that it is in no danger of dying. It seems therefore, that Keen's despair is unfounded and that information systems have indeed grown into a separate, identifiable discipline, even if the field is best described as a 'fragmented adhocracy' ([3]).

The existence of an established scientific community in information systems has been given formal recognition by the recent formation of the Association for Information Systems, a professional society in the tradition of scientific societies, with 1400 members in 35 countries. A recently compiled directory of information systems academics contains entries on some 4,500 researchers from more than 1,000 institutions. A number of basic University and other curricula for information systems education have been published over the years [2, 6, 18]. The most recent of these is Curriculum '95, a joint effort by the ACM, AIS, DPMA, IAIM and ICIS [10, 7]. The most popular discussion group on the Internet (ISWorldNet) devoted exclusively to information systems matters has a membership which in 1997 approached 1829 from 53 countries [14]. A well-defined scientific community therefore exists.

In addition, if the existence of sound academic scholarship is further testimony to the existence of a 'discipline', then information systems can proudly point towards a dramatic growth over the past three decades in the number of scientific journals reporting on research in this area [12]. An even more recent study on research outlets showed that, amongst twenty-seven established journals carrying articles in this field, at least three of the most highly rated top ten are devoted exclusively to the discipline.

Yet, can it be said that the information systems discipline has been conclusively defined and that the research problems and research methodologies prescribed for it have been accepted by all who consider themselves to be working in this field? A re-examination and extension of an earlier (1988) list of keywords for use in classifying information systems literature [4] includes a list of the reference disciplines of information systems, as well as lists of the external environment, the technology, the organisational environment, etc., of information systems. We could argue that this very comprehensive list of keywords (nearly 1300) and other classifications define and describe the discipline of information systems accurately and usefully. For instance, the reference disciplines were listed as:

behavioural science, computer science, decision theory, information theory, organisation theory, management theory, language theories, systems theory, research, social science, management science, artificial intelligence, economic theory, ergonomics, political science, psychology. This list reflects the interdisciplinary or pluralistic nature of information systems.

In the same vein, [19] did a study on the themes of submissions to the journal Information Systems Research and produced a list of keywords, concepts and associations that characterise the categories into which they grouped the research questions of articles submitted. This list demonstrates conclusively that the subareas of the discipline (organisational, behavioural and managerial issues) are well established and attract a large number of researchers on a long-term basis. Swanson & Ramiller conclude by observing that the discipline still exhibits the 'fragmented adhocracy' identified by Banville & Landry, and is still topically diverse and '...based on appeals to significantly different and partly incommensurate reference disciplines'.

Thus, fragmentation can have adverse effects – something that information systems researchers should be aware of. However, fragmentation of the discipline of information systems may be evident in the field for a very long time. As has been pointed out [17, 8], the discipline as a whole follows trends in information technology, and researchers tend to build their interests around new technology (e.g. the earlier interest in expert systems and decision support systems, and current interest in computer-supported co-operative work). As information technology evolves, so the research interests will follow these new directions. Although we may wish it were different, it remains a fact that information technology is still a major reference discipline of information systems, and will remain so as long as researchers struggle to separate the fundamental or common issues in different fields from the technological ones.

Clearly, then, information systems is internationally well-established as a flourishing discipline. In the Southern African context it is important that the discipline should not merely flourish but be seen to flourish. To this end, this editorial calls on academics and especially on practitioners to add your contributions, via a submission to SACJ.

References

[1] M. Alavi and P. Carlson. A review of mis research and disciplinary development. *Journal of Management Information Systems*, 8(4):45–62, 1992.

[2] R.L Ashenhurst. Curriculum recommendations for graduate professional programs in information systems. *Communications of the ACM*, 15(5):364–398, 1972.

[3] C. Banville and M. Landry. Can the field of mis be disciplined? *Communications of the ACM*, 32(1):48–60, 1989.

[4] H. Barki, S. Rivard, and J. Talbot. A keyword classification scheme for is research literature: An update. *MIS Quarterly*, pages 209–226, June 1993.

[5] R.B. Cooper and M. Blair, D.and Pao. Communicating mis research: A citation study of journal influence. *Information Processing & Management*, 29(1):113–127, 1993.

[6] J.L. Couger. Curriculum recommendations for undergraduate programs in information systems. *Communications of the ACM*, 16(12):727–749, 1973.

[7] J.L. et al. Couger. Guidelines for undergraduate is curriculum. *MIS Quarterly*, 19(3):341–360, 1995.

[8] P. Ein-Dor and E. Segev. A classification of information systems: Analysis and interpretation. *Information Systems Research*, 4(2):166–204, 1993.

[9] S.B. Eom and S.M. Lee. Leading us universities and most influential contributors in decision support systems research (1971-1989). *Decision Support Systems*, 9:237–244, 1993.

[10] J.T. et al. Gorgone. Information systems '95 curriculum model — a collaborative effort. *Data Base*, 25(4):5–8, 1994.

[11] R.A. Hirschheim, J. Iivari, and H. Klein. An assumption analysis of the five emergent approaches to information systems development (isd): A contribution to the paradigmatic foundations of alternative approaches to isd. Working paper, Binghamton, 1995.

[12] C.W. Holsapple, L.E. Johnson, H. Manakyan, and J. Tanner. Business computing research journals: A normalized citation analysis. *Journal of Management Information Systems*, 11(1):131–140, 1994.

[13] J. Iivari. A paradigmatic analysis of contemporary schools of is development. *European Journal of Information Systems*, 1(4):249–272, 1991.

[14] B. Ives. Posting to `listserv.hea.ieon` 21 July 1997.

[15] P.G.W. Keen. Mis research: Reference disciplines and a cumulative tradition. In *Proceedings of the First International Conference on Information Systems, Philadelphia*, pages 9–18, 1980.

[16] P.G.W. Keen. Relevance and rigor in information systems research: Improving quality, confidence, cohesion and impact. In H-E. Nissen, H.K. Klein, and R.A. Hirschheim, editors, *Information Systems Research: Contemporary Approaches & Emergent Traditions*. North-Holland, Amsterdam, 1991.

[17] K. Lyytinen. Information systems and critical theory. In M. Alvesson and H. Willmott, editors, *Critical Management Studies*. Sage Publications, London, 1992.

[18] J. Nunamaker, J.L. Couger, and G.B. Davis. Information systems curriculum recommendations for the 80s: Undergraduate and graduate programs. *Communications of the ACM*, 25(11):781–805, 1982.

[19] E.B. Swanson and N.C. Ramiller. Information systems research thematics: Submissions to a new journal, 1987-1992. *Information Systems Research*, 4(4):299–330, 1993.

Recursive Specifications and Formal Logic
What Benefits For Intelligent Tutoring Systems?

Lot Tcheeko

Department of Computer Science, High School of Polytechnic, P.O. Box 8390, Yaounde Cameroon, ltcheeko@polytech.uninet.cm

Abstract

The pedagogical assessment of a tutoring system relies upon a proof of convergence: for such a tutor the correction of student mistakes must not forever delay the teaching process. Such a tutor must provide uniform diagnosis in order to drive the dialog with the student. Student learning can be simulated by a compilation of knowledge, but it is also necessary to compile knowledge for the teaching process. How can the tutor adjust these levels of compilation while keeping uniform diagnosis? We propose here to use recursive specifications, which consists of formalizing the definition of a class of problems at the same time as their solutions. Such a class "doesn't hide information": this allows its subsequent compilation.

Keywords: *artificial intelligence, tutoring system, pragmatic approach, computational models, recursive specifications, student model*

Computing Review Categories: *F.m, H.5.3, K.3*

1 Introduction

In this paper, we consider the theoretical foundation for implementation of efficient and intelligent tutors to teach programming. To learn to construct programs implies to understand semantics of programs, therefore the different logics underlying program semantics may be useful to represent knowledge in the studied domain. However, designing intelligent tutors causes many other problems concerning diagnosis, explanations and student guidance.

This paper is structured in the following way. In section 2 we explain how the recursive specification method may be a good support for solving most of these tutorial problems. Section 3 presents the notion of recursive specification and how it allows the use of the domain structure to produce explanations. Furthermore, a survey of related work is proposed.

2 Intelligent Tutors and Teaching how to Program

CAL (Computer Aided Learning) software can be interpreted in terms of two families of process:

- The process driven by diagnosis or the learning process, for example the opportunistic intervention of the tutor in a learning environment. These processes provide the correction of mistakes and also student guidance. They play a leading part in user help systems.
- The process driven by the pedagogical strategies, which we shall design by teaching process, are preponderant in the actual tutoring system (LISP-Tutor [6], MHO [7], ...).

In a real implementation, interaction between the two processes is needed: a significant function of the pedagogical software is their synchronization. In a classical tutor, the steps of the two processes are run alternatively and it is possible to merge them. For an intelligent tutor, it is not the case because the student model, allows a more versatile synchronization.

The epistemic logics (intended for formalizing knowledge and knowledge acquisition) and the doxastic logics (intended for formalizing belief) make possible the formalization of the reasoning and provide epistemic specifications and external descriptions, in terms of finality, of the methods for solving the specified problem. Their application to the student cognitive modelling seems to us questionnable or, at least, difficult. Instead, the same logics may be used to make the knowledge explicit, procedurally imbedded inside a program, and ease the generation of guidance and explanations.

A pragmatic student representation must keep a qualitative diagnosis and a specific context. The diagnosis and the specifications spread along the structure of the field to be taught: when a pedagogical plan is carried out by the tutor, the preconditions for the execution of an exercise are read from the student model. Items to store are generalization of the results of the diagnosis programs. The relevant informations to be registered are then diagnosis classes to which a tutorial context is added. As diagnosis must be generalized and classified, one is lead to reorganize the tutorial expertise in two levels of knowledge according to a distinction parallel to the one between specific model and general model for the student representation

The link between the qualitative, general model and the specific model is important, not only to produce quali-

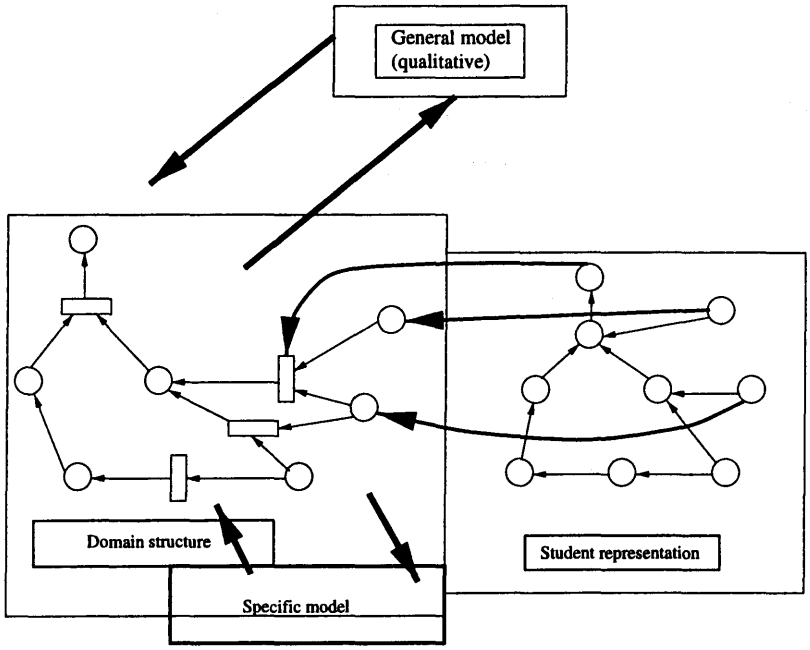

Figure 1:

tative diagnosis but also to generate explanations.

This organization is well applying to a pragmatic architecture: the derivation at the level of the proofs corresponds to the construction of a specific model, while the general model and the qualitative reasoning corresponds to the abstraction at the level of the heuristics. The dichotomy specification/program enables to define a space of problems, that is to use the domain structure to describe objective and resolution plans. In a dual way, it enables to refine specification and then with respect to the student reactions to refine the given problem.

3 Recursive Specification

In this pragmatic architecture, there is a basic dichotomy between specification and implementation, and more generally between problem and solution. Usually intelligent tutoring systems for programming use program patterns called plans: diagnosis is ensured by matching these patterns.

Broadly, program synthesis makes use of two kinds of techniques: the logic approach and the one using programs transformations. Let us consider the logic approach: there are several logic formalisms. For instance, for classical logic, the meaning of a sentence is externally defined by a collection of interpretations. For intuitionistic logic, the meaning of a sentence is internally defined by calculus rules. The difference between both points of view matches to the one between denotational semantics and operational semantics for programming. For CAL, one must set the student a problem by an external description, but one also must use the internal approach to explain its construction and diagnose student's solutions.

The introspection and reflexivity properties of the domain expert seem to us significant. In order to be able to reason about the a priori wrong attempts of solutions from a student, the expert must first be able to think about his own reasoning. The student modelling is a secondary task of the tutor: an accurate diagnosis is useful only for the pedagogical strategy of the tutor.

Ohlssom[10] suggests that the student model is used to anticipate if a pedagogical plan applies. It is an "oracle" forecasting student behaviors and it can also be called runnable student model (the model can in some cases stay undecided and answer "perhaps" to the question "will the student do that?").

Clancey[3] calls the general model an abstract representation which is common to every domain problem and calls qualitative reasoning the inferences into this general model. In opposition to the general model, the resolution of an actual problem uses a specific model.

Gilmore [6] defines the use of the student model as a "learning companion". The ideas for runnable model and learning companion are complementary. When a pedagogical plan is carried out by the tutor, the preconditions for the execution of an exercise are read from the student model. Items to store are generalizations of the results of the diagnosis programs. The relevant information to be registered are then diagnosis classes to which a tutorial context is added. As diagnosis must be generalized and classified, one is led to reorganize the tutorial expertise in two levels of knowledge according to a distinction parralel to the one between specific model and general model for the student representation.

Mizzaro[9] proposes a formalism that can be used to represent the meaning of utterances at a semantic/pragmatic level. The recursive model he defined as an instance of the class of computable models is seen as an

abstract data type whose formal specifications are given and used for modeling the semantics of natural language utterances.

The paradigm called "specification as type" considers a type (a sentence) as a program specification, and views a term (a proof) as a program. Such a program is intended for verifying its specification. In other words, a type is a sentence viewed as the set of its proofs. This leads to numerous applications for defining program semantics.

However, a specification as type gives only an internal syntactic description of what the program is doing, not why it is doing it. The tutoring system needs some kind of external description to relate programs with one another. Moreover, intuitionistic logic is constructive, but classical logic is better able to describle a program structure, and to give external descriptions for a program.

We shall extend the notions of specifications as a type to deal with program transformations. For instance, a tutor must be able to refine specifications. The sentence

$$\forall x : N \leftarrow [1,n].\exists \sigma : [1,n] \leftarrow [1,n].\forall i : [1,n].\forall j : [1,n].$$

$$(i = j \Leftarrow \sigma(j)) \wedge (x_{\sigma(i)} \leq x_{\sigma(j)} \Leftarrow i \leq j)$$

is not sufficiently precise as regards to the numerous sorting algorithms. However, it describes the goal of all sorting programs. It is an invariant for all possible algorithms and tells what the program has to do.

To define natural integers, the scheme:
$(X(x) \Leftarrow x = 0) \wedge (X(x) \Leftarrow (\exists y.x = succ(y)))$ is not adequate, because the witness of $\exists y$ must be realised, or computed, and so the induction scheme has an algorithmic content. The induction rule is the usual one.

Instead, the scheme $(X(x) \Leftarrow x = 0) \wedge (X(x) \Leftarrow (pred(x) : X))$ is adequate, the predecessor pred is enough to define recursion, as equality $(x = 0)$ and the typing `pred(x) :X` have no algorithmic contents.

The usual induction rule mask information necessary to the search for a proof: it is the "inventor paradox". To prove a sentence by induction, one has to guess a stronger induction hypothesis. In fact, the induction rule is only an approximation of ideal induction, with an infinity of premises, corresponding to the intended semantics for universal quantification. The usual induction rule corresponds to iteration and does not enable the recursive definition of Skolem's function. A solution for this problem was found by Feferman [4], and called "classes inductive generation". Here a class is an inductive set of proofs, defined as the least fixed point for an induction scheme, and such that the associated induction rule corresponds to a recursion operator.

Program transformations are a standard specification and programming method for PROLOG. These transformations are folding and unfolding. In pure PROLOG, an assertion is a formula which can be checked by unification.

For instance:

$list([])$
$List(\_|q|]) \leftarrow list(q).$

define a class, named List. Indeed, this scheme use only unification and a recursive call. The predecessor calculus is also done by unification: this predecessor is the selector tail. It suits well to the idea of solving a simpler problem.

A more complex class, for a more complex formula without algorithmic contents, is the class difference List:

$\exists u : List.\exists v : List.reverse(u,v), append(v,y,x)$

which induction rule justifies the program:

$reverse(x,y) \leftarrow reverse/3(x,[],y).$
$reverse/3([],y,y).$
$reverse/3([t|q];x,y) \leftarrow reverse/3(q,[t|x],y).$

In the preceding paragraphs, we limited the discussion to the use of one variable predicate X in an induction scheme, to eimplify notation, but the same results hold for simultaneous recursive definitions. One can describe predefined (or compiled) predicates with assertions.

SMALLTALK classes are inductively defined: the class message allInstance returns the partial extension of the receiver class, that is the set of already made objects, while instantiation messages such as new, resume the generation process. Finding an object class is like giving a name to the process which generates it. Finding this class is an assertion, a feature of the object without algorithmic content. Then, the notion of class as type without hidden information corresponds well to SMALLTALK and this fact goes against the use of multiple inheritance in the field of tutoring systems. The encapsulation of contexts into SMALLTALK objects, play the part of the closure operator(cf annex).

The use of recursive specifications for CAL is to define a class of problems and their solutions at the same time to allow a uniform diagnosis. This compilation enables simulation of student learning and also refining of teaching goals.

4 Application

Compiling ability is especialy interesting for CAL, in order to adapt explanations to the student's level of knowledge. Some intelligent tutors simulate the student's learning process by a compilation. This needs an operator to kill the algorithmic contents of a program by transforming its specification into a simple hypothesis. In fact, the specification concept used in a tutor is a partial specification. To sum up, a program has three kinds of knowledge:

- **Tautologies** According to the formalism the knowledge actually corresponds to tautologies or translate the meaning of the logical connectives. They do not affect the realization of the final program. It is then possible to integrate assertions into them, these assertions correspond to hypotheses during the program deriva-

tion and to mistakes when executed.

- **Proofs** They are knowledge used to verify that an inference is sound and alowing the verification of the final problem according to his specification. They are in fact partial proofs, or hypothetical proofs in the presence of assertions.
- **Heuristics** They are knowledge allowing to guide and control the search of a proof and then the building of the final program. As the assertions escape the normal control, they allow the application of heuristics.

From a didactic point of view, the knowledge which we have called proofs correspond to the explanations whereas the heuristics correspond to the student guidance. Presenting heuristic knowledge allows to guide the student in the resolution of the problem.

From a tutor point of view, a resolution method corresponds to the following diagram:

The resolution mode makes use of explicit knowledge which are invariant during the derivation process and of implicit knowledge in the interpreter, obviously independant from the running program.

The diagram is different from the student point of view:

If the semanticcs of the domain to be taught is rich enough and capable of various representations, the tutor cannot make a preview on the conceptual representation built by the student. The only way to do the diagnosis is to use the common part: the interpretation. To be able to explain to the student the methods used, (a) the interpreter must be able to explain its own behavior, (b) the interpreter must be able to explain the use of the heuristic knowledge. At this point, it is very attractive to merge both functions in the interpreter itself. Even if a tutoring system doesn't build up natural language problem statements and if this statements are static records written by the author, the system must use an internal representation of these problems to be able to introduce them and their relations with one another. One would expect that this representation should not be more complex than the representation of solutions. With an incomplete specification, the tutor can steer the student to understand a class of problems. It may use examples, counter-examples and successive refinements.

The author of a teaching system, designed to use the domain structure, takes a problem (or exercise) as data to build up other more complex problems, or to state that this problem is a particular instance of another. The same is true for a tutoring system able to synthesize problems, or simply, to build up derived problems (such as verifying or correcting).

Instead, a problem can be thought of as a program to diagnose a student's tentative solution. If the initial problem splits one can try to split the potential solution and delegate the diagnosis of each piece to the matching subproblem. Our basic idea is that explanations and diagnosis propagate along the domain structure.

One point of our work, detailed in [11], is to apply these ideas to teach programming. We use a formal derivation of solutions from problems, and we dynamically build derived problems.

5 Conclusions

We have shown the interest of recursive specifications in tutoring systems. They enable the definition of classes of problems and their solutions, and at the same time, the relevant diagnosis programs. We also pointed out the need to extend first order logic to the second order, to solve some problems in CAL. In spite of very different formalisms, the needed extension corresponds to the one used by intensional logic, that is a modal and self-referential formalism. We found SMALLTALK a relevant programming language to do this.

Extension of this intentional logic to stronger self-referential abilities makes the domain expert to behave as a qualitative, general model. This abstract model uses meta-knowledge, either operationally, as control heuristics, or declaratively, as meta-theorems about the expert provability. The dichotomy specification/program enables to define a space of prob lems, that is to use the domain structure to describe objective and resolutions plans. In a dual way, it enables to refine specification, and then with respect to the student reactions, to refine the given problem.

The difficulty with applying these ideas to student modelling points out that the concept of knowledge is still causing problems. In the context of tutoring systems, such a problem is the epistemic status of objects which are topics of discourse. This status fluctuates between a platonistic view, where these objects exist independently of the tutoring discourse (the student is already acquainted with them), and an intuitionistic view, where they represent the result of a mental construction (the student can only grasp some "know-how" about them). A good balance between both approaches must be a fundamental aim in designing an intelligent tutoring system.

References

[1] Joyce M. Ivill Alan M. Lesgold, Jeffrey G. Bonar and Andrew Bowen. An intelligent tutoring system for electronics troubleshooting. Technical report, Learning Research and Development Center, Univ. Pittsburgh, Pittsburgh Pe, 1985.

[2] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 1:1–7, 1985.

[3] William J. Clancey. The role of qualitative models in instruction. In John Self, editor, *Artificial Intelligence and human learning*, pages 49–68, London, New york, 1988. Chapman & Hall.

[4] Solomon Feferman. A language and axioms for explicit mathematics. In *Lecture Notes in Mathematics*, volume 450, pages 87–139. Springer, 1975.

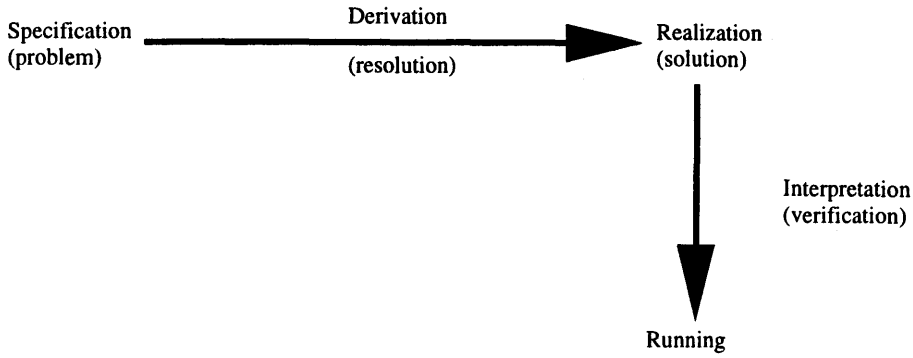

Figure 2:

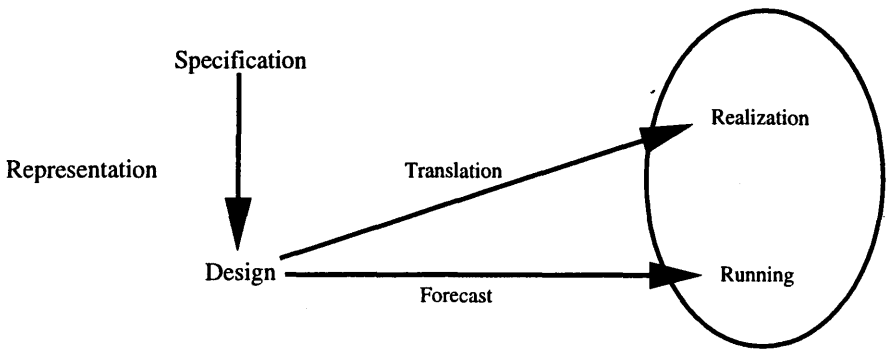

Figure 3:

[5] David Gilmore and John Self. The application of machine learning to intelligent tutoring systems. In John Self, editor, *Artificial intelligence and human learning*, pages 179–196, London, New-York, 1988. Chapman & Hall.

[6] Brian J. Reiser John R. Anderson, C. Franklin Boyle. The geometry tutor. In *Proceedings of the Xth International Journal on computing and artificial Intelligence, Los Angeles Ca*, volume 1, pages 1–7, 1985.

[7] Alan M. Lesgold. Going from intelligent tutors to tools for learning. In *Second International Conference on Intelligent Tutoring Systems, Montreal, Canada*, June 1992.

[8] Stefano Mizzaro. Towards recursive models — a computational formalism for the semantics of temporal presuppositions and counterfactuals in natural language. *Informatica*, 21:59–77, 1997.

[9] Stellan Ohlssom. Some principle of intelligent tutoring. *Instructional Science*, 14:293–326, 1986.

[10] W. L. Johnson & Soloway. Intention based diagnosis of programming errors. In *Proc. National Conference on Artificial Intelligence, Austin TX*, volume 1, pages 162–168, 1984.

[11] Lot Tcheeko. Une approche contextuelle de l'aide au dignostic d'erreurs en langage machine. In *Congres Applica '90, Lille, France*, September 1990.

A Implementation of Cut and Consult

Cut: assoc
"Succeed always, exit the containing method during backtracking"
| blk mark |
assoc key size = 1 ifFalse :[^self].
mark := unboundVars size.
blk := assoc value
[(blk := blk value) isContext] while true:
seft unbindVar: mark.
Assoc key head value

Consult: assoc
"Ask the second argument as the receiver (a Smalltalk expression) to solve the goals contains in the first argument, e.g. consult(father(x, 'John'), family new)."
| blk |
assoc key size = 2 ifFalse :[^self]
blk := self first :assoc.
Blk class = = Relation ifFalse: [self error : 'first argument must be predicate'].
^[(self second: assoc) value perform: (blk head, ' :') asSymbol
with : (Association key : blk tail
value : assoc value)]

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications of Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript
Manuscripts for *review* should be prepared according to the following guidelines:

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
- the title (as brief as possible)
- the author's initials and surname
- the author's affiliation and address
- an abstract of less than 200 words
- an appropriate keyword list
- a list of relevant Computing Review Categories
- Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text according to the Harvard. References should also be according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page,

`http://www.cs.up.ac.za/sacj`

which gives a number of examples.

SACJ is produced using the LaTeX document preparation system, in particular LaTeX 2_ε. Previous versions were produced using a style file for a much older version of LaTeX, which is no longer supported. Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

Charges
Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

These charges may be waived upon request of the author and the discretion of the editor.

Proofs
Proofs of accepted papers may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Letters and Communications
Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words. Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book Reviews
Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement
Placement of advertisements at R1000.00 per full page per issue and R500.00 per half page per issue will be considered. These charges exclude specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 21, August 1998
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 21, Augustus 1998
ISSN 1015-7999

Contents