

SICS Technical Report
T92:09

ISRN : SICS-T—92/09-SE
ISSN : 1100-3154

Applications of Partial Inductive Definitions

Conclusions from some projects run at SICS KBS-lab 1986-1992

by

Per Kreuger

April 1992

Swedish Institute of Computer Science
Box 1263, S-164 29 KISTA, SWEDEN

Applications of Partial Inductive Definitions

Conclusions from some projects run at SICS KBS-lab
1986-1992

by
Per Kreuger

piak@sics.se
Swedish Institute of Computer Science
Box 1263
S-164 28 Kista
Sweden
April 1992

Keywords: Knowledge Based Systems, Logic Programming, Non monotonic reasoning, The programming language GCLA.

Abstract:

This paper describes the history of and some conclusions from a series of projects run at SICS KBS-lab during the period from 1986 to 1992. The projects have in common that they all are applications of the theory of partial inductive definitions.

Table of Contents

1	Motivation of the Research	1
2	Background	2
2.1	Theoretical Background	2
2.2	Programming language (KBS tool)	2
2.3	Program verification/synthesis and the proof editor Pi	3
3	Project History	4
4	Results	6
4.1	Published papers and Conference Contributions	6
4.2	Academic degrees	7
4.3	Workshops	7
4.4	Applications	7
4.5	International Contacts	8
4.5.1	The ESPRIT working group: GENTZEN	8
4.5.2	Others	9
5	Conclusions and Evaluation	9
6	Future Work	10
	Bibliography	11

1 Motivation of the Research

The goal of Knowledge Based Systems (KBS) technology is to develop methods and tools for development and maintenance of KBS. In order to build systems based on knowledge we need a formalism to *represent knowledge* and (preferably) well understood algorithms operating on these representations. These algorithms should model (significant parts of) the kind of *reasoning* that humans do from their acquired knowledge.

State-of-the-art approaches views KBS development mainly as a modelling activity. Most modern approaches also support product management, reusability and integration with standard software development techniques.

Key issues in KBS concerns the specification and development of the models necessary for the application at hand. In recent years *the expertise model* has become predominant in KBS development technology. This model is defined at the knowledge level and typically relies on domain, inference, task and problem solving knowledge to model the reasoning of an intelligent agent (the expert). As a part of the development of a KBS this knowledge level model is transformed into a symbolic level model that can be manipulated by a computer.

To support development and testing of the different models we need high level languages and tools. Languages for development of executable specifications and for implementation of different reasoning mechanisms at a high level of abstraction are highly desirable in this context.

One approach to these problems involves using logic formalisms and inference mechanism developed in mathematical logic and proof theory. One could certainly say that the programming language Prolog arose in this context of trying to find general problem solving and reasoning mechanisms.

Now, compared to the systems traditionally studied in logic, Prolog is a very weak system. There is no inherent concept of negation, nor of evaluable functions; the concept of equality is the simplest imaginable etc. In practice however, and for general programming, Prolog has indeed proved to be a very useful tool. This is because Prolog makes a successful compromise between language expressiveness and computational efficiency.

Prolog has been successfully used for implementing KBS for a long time, but there are some recurrent problems with this approach. Frequently when developing a KBS the knowledge engineer needs to try out his formalizations in practice i.e. he wants to develop some kind of prototype system. In this case the weakness of the implementation language forces the knowledge engineer to translate his high level concepts into low level Prolog code at a too early stage in the development process, thus obscuring the essence of the formalization behind insignificant technicalities and idiosyncratic encodings.

From these facts arises the need for a higher level modelling tool: A language with reasonable computational efficiency but expressive enough to not deter the knowledge engineer from the essentials of the abstractions inherent in the problem formalization itself. Of course, we cannot expect to find such a language in a first attempt. The research reported in this paper simply points out some directions and results in this important field.

2 Background

2.1 Theoretical Background

When it comes to the choice of a logic formalism the most common is First Order Predicate Calculus (FOPC). There are some problems even with this classical and very powerful logic. Some types of reasoning require e.g. that we make decisions based on incomplete knowledge or on account of what we do *not* know. Logics used in this context are various modal or non-monotonic extensions of FOPC. The choice largely depends on the application.

One problem with both these approaches is that there does not exist sufficiently efficient execution mechanisms for these systems. Another approach is to extend Prolog or some similar system to handle the constructs that are needed to formalize the problem domain. This is usually done either within the context of FOPC or some extension of FOPC.

The approach we have taken at SICS KBS lab differs slightly from these in that we have chosen to extend Horn Clauses (Prolog) within a formal system called the theory of Partial Inductive Definitions (PID). The reason for this choice is that this theory seems to be particularly suited to do non monotonic and default reasoning. The theory is more low level than traditional logic systems in the sense that it is a system in which we can define a logic specially tailored to our problem domain, and then formalize the domain in that logic. The theory of PID was developed at SICS in the early stages of the project and based on work done by Lars Hallnäs before he was employed at SICS. For references to the theoretical foundations of PID see [Hal 91a, Hal 91b, Hal 92a]. For its role as a basis for GCLA and its relation to Logic Programming see [HSH 90]. For background material on Inductive definitions see [Acz 77], and on Logic Programming see [Rob 65, Ll 87, Kow 79b, SS 86].

The work on PID at SICS has gone in three major directions:

- The development of the theory itself and theoretical results describing different properties of the formalism. This work is now largely completed.
- The development of a programming language GCLA to be used as a KBS modelling tool. GCLA is significantly weaker than the more complete finitary formalism of PIDs implemented in the proof editor Pi (discussed below). GCLA stands in a relation to the theory of PIDs as does Prolog to FOPC.
- Work on applications of the theory in the field of program verification and synthesis and lately in the development of a semi automatic proof editor (named Pi) for general logics.

2.2 Programming language (KBS tool)

GCLA (Generalized Horn Clause LAnguage) is a programming language built on a generalization of Horn clause logic (i.e. the “logic” of Prolog). Although the language allows both functional and relational styles of programming the language is perhaps best described as a logic programming language.

We generalize Horn clauses by allowing assumptions in the bodies of clauses. This means e.g. that $(A \rightarrow B) \rightarrow C$ can be seen as a clause. The semantics of this generalization is not given in the traditional way (see e.g. [Ll 87]) as a somewhat larger subset (than Horn clauses) of first order predicate Calculus (FOPC), but rather in terms of a more general (low level) theory, i.e. the theory of PID [Hal 91a].

The interpretation of a program in this theory do not give us a fixed logical reading of the program, rather the program itself (in a well defined way) generates a logic in which inferences can be made. This means e.g. that the arrow constructor does not always correspond to logical implication.

On the other hand the resulting formalism is strong enough to give us a very useful form of negation which is sound for all programs that *do* allow a logical reading and can give bindings for free variables in negative queries. In fact the language implements the Closed World Assumption (CWA) without resorting to Negation as Failure (NAF). The language also allows us to pose hypothetical questions to a database, to do default reasoning and functional programming in a very natural way. The language can possibly even be used to do Object Oriented Programming (OOP), even though this field still needs more investigation.

The execution mechanism of the language is built around a set of inference rules and proof search heuristics. The computation is initiated with a query to the interpreter in the form of a sequent " $A \multimap C$ " and continues with a search for a proof of some instance " $A\sigma \multimap C\sigma$ " of the original sequent. As in Prolog the substitution σ is generally regarded as the result of the computation. The main difference from Prolog here is the presence of the assumptions A .

For theoretical work on GCLA see [Hal 91a, HSH 90, Kre 92, Aro 92b]. For a description of the language itself and examples of its application see [GCLA 90, Kre 92, Aro 89c, Aro 91a, Aro 91b, Aro 92a]. For implementation issues see [Aro 89a, Aro 89b, Aro 92b].

2.3 Program verification/synthesis and the proof editor Pi

An important part of KBS technology is reasoning, the process of deriving new knowledge from old knowledge. Knowledge can conveniently be represented as formulae of some logical system. The reasoning process can then manipulate the logical formulae in a precise way.

It turns out that many reasoning mechanisms can be expressed using PIDs in a natural way. In fact, the theory of PIDs can be used as a general logic. A general logic is a formal system that can emulate other logics ("object logics"). The advantage is that theoretical results and computer programs (e.g. a reasoning program) developed for use with the general logic can be used with all the object logics as well.

To represent an object logic using PID, the semantics of the logic in question is defined as a PID. In a sense, the definitional clauses encodes the truth (valuation) functions of the logic. The formal system of PID then behaves as a proof system for the logic in question.

In its naive form, this approach is restricted to truth-functional logics. However, non-truth functional logics such as modal logics, can be handled in a natural way by introducing truth-functional judgements. For the case of modal logics with Kripke semantics, such a judgement is "the formula A is true in world \mathcal{W} ".

The idea of using PID to represent logics first appeared in [Hal 87b]. Some other early work is described in [EH 88]. A detailed account will be given in [Eri 92b].

In the previous section, we remarked that a GCLA program itself "generates" a logic, in which inferences are made. The use of PID as a general logic is an essential use of this property.

To make practical use of PID as a general logic, an interactive derivation editor, Pi, has been developed. Using Pi, a proof in the formal system of PID can be built up interactively. The user is presented with a proof tree in graphical form on the display and manipulates the proof by issuing commands with the mouse. The principles of Pi will be described in detail in [Eri 92b]. [Eri 92a] is a manual for the system with some explanation of the underlying principles. The structure of the user interface of Pi is inspired by the IPE proof editor system [RT 88].

Since a particular PID is, in general, an infinite mathematical object, the use of PIDs on a computer in a well-defined way poses a problem. That problem was addressed in [Eri 91], by the definition of a finitary version of PID and the associated formal system. This finitary system is a superset of the pure GCLA language, and is used to represent PIDs in Pi.

One particular application of the use of PID as a general logic is the definition of a framework for the verification of pure GCLA programs (and thus also of pure Prolog programs). Within this framework, a PID is given defining the semantics of some specification language. The GCLA program to be verified can then be thought of as generating a model for that language. If an intended specification can be proven to hold using the generated proof system, we know that the specification is true in the model generated by the program. This can be shown to imply that the program is correct according to the specification. This work is described in [EH 88] and [Eri 92b].

3 Project History

In 1986 Lars Hallnäs was employed at SICS to conduct research on logic formalisms for knowledge representation (KR) and on reasoning mechanisms for non monotonic reasoning. Shortly after that two undergraduate students: Martin Aronsson and Johan Montelius were employed part time to do an experimental implementation of the ideas that Lars Hallnäs had been working on.

During this period Lars Hallnäs produced a series of drafts on a formal system of inductive definitions, the first publicly available being [Hal 86]. Martin Aronsson and Johan Montelius completed a number of interpreters for a programming language based on the theory of inductive definitions under development by Lars Hallnäs.

The basic ideas of a Generalized Horn Clause Language based on a theory of PIDs was first presented at the Fourth Japanese-Swedish workshop on Fifth Generation Computer Systems at Skokloster July 1986 [JSW 86].

During the end of 1986 and the beginning of 1987 Lars Hallnäs worked with possible applications of the theory and the formal language based on it. He explored the strength of the formalism in the domain of default reasoning and published a paper on the subject in the 1987 conference on the Frame Problem in Artificial Intelligence [Hal 87a]. Together with Martin Aronsson he also produced a set of examples in the domains of non monotonic reasoning, planning and logic programming with negation. By this time Johan Montelius had left the emerging group.

Later on it was also discovered that the language could be used as a functional language as well as a relational (logic programming) language. The integration of these methodologies however was not investigated until later (see [Aro 91a]).

Also involved at this early stage was Peter Schroeder-Heister, then at the University of Konstanz, BRD. Lars Hallnäs and Peter Schroeder-Heister initiated a cooperation during 1987 aimed at describing the theoretical foundations of a Generalized Horn Clause Lan-

guage. This cooperation have so far resulted in two papers written jointly by Lars Hallnäs and Peter Schroeder-Heister [HSH 90].

During 1987 Lars-Henrik Eriksson developed the framework for verification of logic (GCLA) programs. This work was presented as his Ph. Lic. thesis during 1988 [EH 88].

Towards the end of 1987 a language then called Generalized Horn Clause Language emerged from the group [AH 88]. This work included a set of programming examples.

Martin Aronsson now commenced work on an abstract machine for the language and in the beginning of 1988 the undergraduate student Peter Olin was employed to assist Martin with the development of a compiler. This worked aimed at showing that the language at least in principle could be efficiently implemented.

During 1988 the group initiated a cooperation with people at Ellementel working with languages for specification of telephone services. The language was not at this early point mature enough for industrial application but the short cooperation was very valuable for the group at SICS who, in Ellementel's application, saw the possibility to use the language as a planning and simulation tool. Martin Aronsson later published a report on the subject [Aro 89c].

These results motivated further research into the use of the language as a planning tool, e.g. it was used for implementing a planning system for construction site time planning and scheduling [Pal 89]. This work was inspired by and later on used in a part of the Swedish MDA project [MDA 90] where Martin Aronsson was also engaged at the time.

In June 1989 Martin Aronsson gave a presentation of the language and some program examples at the SCAI conference held at the University of Tampere, Finland.

Towards the end of 1988 Lars Hallnäs left SICS and joined the Programming Methodology Group at Chalmers. He continued as a part time supervisor for Martin Aronsson and Lars-Henrik Eriksson and later for Per Kreuger who began working with the group by the end of 1989.

During the end of 1989 the work on the abstract machine was completed and was published in two SICS reports [Aro 89a, Aro 89b]. Shortly after that the compiler was also complete. After that Peter Olin left the group.

A first version of the Pi proof editor was completed early in 1990. Since then, continued work on Pi, on the principles for using PID for general logic, and on the theory of the finitary version of PID used by Pi has been done simultaneously.

At this time it was becoming more and more clear that the language (by now renamed GCLA) introduced control problems on a completely different scale than e.g. Prolog. The execution mechanism of GCLA at this time introduced a vast number of choices for non trivial applications and had insufficient means for guiding and controlling search. There had been discussions within the group on this problem for some time, but at this point it was realized that the language could not really be used for real life application without a more flexible and powerful control mechanism.

In the beginning of 1990 Per Kreuger, who had been working at SICS with the use of Higher Order Logic programming languages for similar applications, was involved in a cooperation with Lars Hallnäs and Martin Aronsson to solve the control problems. This cooperation resulted in a new incarnation of the language called GCLA II with vastly improved control features. This work was published as a part of the survey paper on

GCLA in [GCLA 91]. GCLA II was later revised and extended by Per Kreuger and published as his Ph. L. thesis [Kre 92].

The control (meta) part of a GCLA II program is completely separate from the declarative (object) part which gives a clear separation between the declarative semantics of the object program and the (operational) control issues i.e. in the spirit of [Kow 79a].

In addition to their role in guiding and controlling search the control strategies of GCLA II made it possible to integrate the different programming methodologies in a general and coherent manner. E.g. now the integration of relational (logic) and functional programming could be tackled [Aro 91a].

During this period two other students of Lars Hallnäs: Daniel Fredholm and Svetozar Serafimovski began an investigation of using PIDs as type system for λ -terms [FS 92]. Both of them are currently working on related subjects at the mathematics department at the University of Stockholm. See e.g. [Fre 90].

In 1991, Lars-Henrik Eriksson completed work on the finitary version of PID. That work was presented in [Eri 91].

A series of international workshops on extensions of logic programming was initiated by P. Schroeder-Heister in 1989. A first workshop was held in Tübingen 1989. The group at SICS organized the second workshop 1991 at SICS and third workshop was organized in Bologna 1992. Proceedings from the two first workshops has been published by Springer Verlag in their series "Lecture notes in Artificial Intelligence". See [WELP 1, WELP 2, WELP 3].

At each of these workshops the group presented their work and gave demonstrations of their implementations and example programs. This series of workshops initiated the international cooperation now formalized within the framework of the ESPRIT working group GENTZEN (see section 4.5.1 below).

4 Results

4.1 Published papers and Conference Contributions

The following list include papers published by members of the group¹ in journals, international workshops and conferences: [Aro 92a, Eri 91, GCLA 90, GCLA 91, Hal 87a, Hal 91a, HSH 90, Kre 92].

The following have been published by Lars Hallnäs after he left SICS: [Hal 87b, Hal 89, Hal 91b, Hal 92a, Hal 92b].

Some of the above have also been published as SICS Reports. Material only published by SICS include: [AH 88, Aro 89a, Aro 89b, Aro 89c, Aro 91a, Aro 91b, EH 88, Eri 92a, Hal 86, JSW 86, Pal 89].

Closely related work published by people outside SICS include [Fre 90, FS 92, Han 92, S-H 91, S-H 92].

¹ While employed at SICS.

4.2 Academic degrees

Lars-Henrik Erikssons work on a Programming Calculus based on PIDs was presented as his thesis for the degree of Ph. Licentiate in 1988 [EH 88] with Lars Hallnäs and Rune Gustavsson as supervisors. Lars-Henrik Erikssons is expected to complete his Ph. D. during 1992 based on his licentiate thesis, the proof editor Pi and the work published in [Eri 91]. The Ph. D. will be presented at the University of Stockholm with Lars Hallnäs as supervisor.

In 1989 Daniel Fredholm completed his thesis for the degree of Ph. Licentiate [Fre 90]. This work is closely related to the work done by Lars Hallnäs who supervised Fredholms work during 1988 while still at SICS and later on at Chalmers.

In 1991 Per Kreuger completed his work on the control mechanisms of GCLA II and presented it as his thesis for the degree of Ph. Licentiate [Kre 92]. The work was supervised by Lars Hallnäs while at SICS, and later on at Chalmers.

Martin Aronsson is expected to complete his Ph. D. during 1993. The thesis will be presented at the University of Stockholm with Lars Hallnäs as supervisor.

4.3 Workshops

As mentioned above Peter Schroeder-Heister at the University of Tübingen initiated a series of workshops begun in 1989 and continued by the group at SICS in 1991. The series, known as Workshop on Extensions of Logic Programing (WELP), was continued by a group at the University of Bologna in 1992 and there will be a continuation in 1993, probably organized within the framework of the ESPRIT working group GENTZEN (see section 4.5.1 below).

The series is dedicated to work on strengthening the expressiveness of Logic Programing Languages both within the framework of classical logic (FOPC) and non classical logics and other formal systems i.e. various subsets of intuitionistic logic, higher order logic programming, linear logic programming, formal theories of inductive definition etc.

Researchers attending the workshop organized by the group at SICS included: Jean Marc Andreoli, ECRC, München, Germany; Harold Boley, DFKI, Kaiserslautern, Germany; Amy Felty INRIA Rocquencourt, France; Lars Hallnäs, Chalmers University of Technology, Göteborg, Sweden; Seif Haridi, SICS, Stockholm, Sweden; Görg Hudelmaier, University of München, Germany; Evelina Lamma and Paula Mello, University of Bologna, Italy; Don Loveland and Gopalan Nadathur, Duke University, Durham, USA; Dale Miller, University of Pennsylvania, Philadelphia, USA, Frank Pfenning, Carnegie Mellon University, Pittsburgh, USA; Peter Schroeder-Heister, University of Tübingen, Germany and others.

4.4 Applications

Apart from the published smaller programming examples a larger application of the planning methodology developed in [Aro 89c] was done as a Master thesis by Johan Palmkvist and presented in [Pal 89]. This work was continued and extended by Martin Aronsson within the Swedish MDA program (in the domain of construction site planning) [MDA 90].

The project was finished in early 1992, but has been recognized as significant by Swedish building industry and union representatives. As a result, the group has received financing for the specification of a continuation project to extend and develop the results from the MDA program. The new project is planned to start early 1993.

The planning mechanisms of MDA have been reimplemented in GCLA II to verify the utility of the new control mechanisms. The system currently plans the construction of a medium sized office building with reasonable response times.

4.5 International Contacts

The ELP workshops have been a very valuable forum for the exposure of the group to the international research community and has made possible a closer cooperation between the group and several other groups doing research in related areas.

4.5.1 The ESPRIT working group: GENTZEN

Currently the most significant cooperation is the Esprit III working group GENTZEN where the Swedish group now plays a significant role. Initially a group consisting of researchers from SICS, Stockholm; Chalmers, Göteborg; the University of Tübingen, the University of St. Andrews, and the University of München applied for a working group called *Extensions of Logic Programming*. The funds applied for were intended to cover travel expenses for scientific cooperation and continuation of the WELP series of workshops. As the group was considered to small, the ESPRIT commission suggested that the group should merge with the working group named *Common foundations of Logic and Functional Programming*. This was done and a joint proposal for a larger working group, now named GENTZEN, was applied for and approved by the commission.

i) Göteborg

Lars Hallnäs at Chalmers University of Technology continues his research into theoretical aspects of PID and foundations of Logic Programming. He will continue to supervise the work of Martin Aronsson, Lars-Henrik Eriksson and Per Kreuger at SICS, as well as other students (working in related areas) at Chalmers and at the *mathematics department of the University of Stockholm*. The funds supplied by Esprit III will make possible a continued and extended cooperation with the group headed by Peter Schroeder-Heister at the University of Tübingen, Germany.

ii) Tübingen

Peter Schroeder-Heister of the University of Tübingen has a long standing cooperation with Lars Hallnäs. Prof. Schroeder-Heister heads a group at Tübingen that is involved in theoretical work on *General Logic and proof theoretic foundations of Logic Programming* and work closely related to that of Lars-Henrik Eriksson at SICS in Stockholm, i.e. *tools and principles of Automated theorem proving*, as well as *program verification and synthesis*.

The group plans to integrate the work done at SICS on GCLA with work done on Linear and Higher order logics by J.-Y. Girard (e.g. [Gir 87]), D. Miller and others (see e.g. [Mil 89, Mil 90]).

iii) St. Andrews

At the University of St. Andrews, Prof. Roy Dyckhoff heads a group working mainly in automated theorem proving and general logics. This work is closely related to that of Lars-Henrik Eriksson in Stockholm. The group considers using GCLA as a development tool in this context.

iv) Kaiserslautern

A group of researchers under *Harold Boley* at DFKI in Kaiserslautern, Germany, have independently used GCLA for applications of *terminological reasoning* within the mechanical engineering domain. When this became known to the group at SICS a cooperation was initiated and have so far resulted in several visits between SICS and DFKI. Work on the use of GCLA in terminological reasoning has been published by *Philipp Hanschke* in [Han 92]. The group is now involved in the GENTZEN working group which formalizes the cooperation.

4.5.2 Others

Apart from the cooperation now formalized through the GENTZEN group there are some other researchers that have taken part in the WELP workshops. There are in the United States several groups working in the area of *Higher Order Logic Programming*. Most of these have at some time worked together with Dale Miller of the University of Pennsylvania, Philadelphia. Dale Miller's work pioneered the introduction of Higher Order Logic constructs into Logic Programming. Prof. Miller has also worked with other orthogonal extensions of Logic Programming concerned with modules, implications and quantification. Dale Miller has attended two of the workshops on ELP [WELP 1, WELP 2] together with colleagues and former collaborators such as *John Hannan*, *Gopalan Nadathur*, *Remo Pareschi* and *Frank Pfenning*. This cooperation, though not formalized in any way, will hopefully continue through the ELP workshop series.

5 Conclusions and Evaluation

The emphasis of the work done by the group at SICS so far have been on problems of a fundamental and theoretical nature. The first couple of years was used to lay the theoretical foundation of the later work. Implementation work during the early period was experimental at best.

After this initial period, time was spent on developing tools based on the earlier and more theoretical work. GCLA and Pi are two such tools that have emerged from the group. The early versions of GCLA showed that more work on the control aspects of the formalisms involved were needed. A third phase in the GCLA project was thus initiated to solve these problems. The result of this third phase is the control mechanism of GCLA II. The theoretical results from this can be used in Pi as well, and perhaps even in other formalisms.

This third phase is now completed and a fourth is just about to begin in which the tools that are now in state where they can actually be used for prototyping real applications.

The work of the group has reached international renown and international cooperation with several research centers have been initiated. Ideas and techniques similar to the central concepts, e.g. the rule of local (definitional) reflection, used in of the formalism of PID have later independently been suggested by well-known researchers in the area.

E.g. *Tierry Coquand* at the Programming Methodology Group at Chalmers working with the Proof Editor/Theorem Prover *Alf* have incorporated techniques that are very similar to the rule of local reflection. Also, in France *J.-Y. Girard* (who is also involved in the GENTZEN group) have independently rediscovered techniques of very similar intent.

Up till now no serious evaluation of the tools have been done. GCLA and Pi should be applied to a number of larger problems thereby giving material for an evaluation of their practical applicability.

Currently (early summer 1992) we are planing two larger development projects done as Masters thesis by four under graduate students, two at SICS, and two at Chalmers. In parallel with this Martin Aronsson will continue his works in the domain of construction site planning, utilizing techniques based on GCLA. Per Kreuger will work on developing programming methodologies for GCLA in KBS, and supervise two of the under graduate students.

6 Future Work

As mentioned in the previous section the main part of the evaluation still remains. Also when the language is used for practical applications, new methodological and perhaps some more fundamental problems will certainly arise. For the immediate future we plan to do extensive testing and evaluation of the techniques and tools we have developed. In a longer perspective more fundamental methodological issues may again become central.

One could imagine several ways to extend the language of GCLA to give an even more expressive language (e.g. with first and/or higher order quantifiers), however we feel that such work will have to wait until the language as it exists today have been extensively tested and evaluated. Such extensions should in any case be motivated by a need from the application developer (i.e. from the need of the knowledge engineer).

Regarding the proof editor Pi, possible future work includes facilities for automatic theorem proving based on the control mechanisms of GCLA II.

Bibliography

Acz 77

Aczel, P.

An introduction to inductive definitions

In Barwise, J., *Handbook of Mathematical Logic*, North Holland 1977.

AH 88

Aronsson, M.; Hallnäs, L.

GCLA Generalized Horn Clauses as a Programming Language

SICS Research Report R88014, Swedish Institute of Computer Science, Stockholm 1988.

Aro 89a

Aronsson, M.

GAM - An Abstract Machine for GCLA

SICS Research Report R89002, Swedish Institute of Computer Science, Stockholm 1989.

Aro 89b

Aronsson, M.

The Instruction Set for the GCLA Abstract Machine

SICS Technical Report T89004, Swedish Institute of Computer Science, Stockholm 1989.

Aro 89c

Aronsson, M.

STRIPS-Like Planning using GCLA

SICS Research Report R89009, Swedish Institute of Computer Science, Stockholm 1989.

Aro 91a

Aronsson, M.

A Definitional approach to the combination of Functional and Relational Programming

SICS Research Report R91:10, Swedish Institute of Computer Science, Stockholm 1991.

Aro 91b

Aronsson, M.

GCLA User's Manual

SICS Technical Report T91:21, Swedish Institute of Computer Science, Stockholm 1991.

An earlier version was published 1989 under the same title as: SICS Technical Report T89012, Swedish Institute of Computer Science, Stockholm 1991.

Aro 92a

Aronsson, M.

Methodology and Programming in GCLA II

In [WELP 2]. Also available as SICS Research Report R92:05, Swedish Institute of Computer Science, Stockholm 1992.

Aro 92b

Aronsson, M.

Implementation Issues in GCLA: A-sufficiency and the Definiens Operation.

Forthcoming SICS report.

EH 88

Eriksson, L-H.; Hallnäs, L.

A Programming Calculus Based on the Theory of Partial Inductive Definitions

SICS Research Report R88013, Swedish Institute of Computer Science, Stockholm 1988.

An earlier version was published as Eriksson's Lic. Dissertation - LUTDX/(TECS-3011), Lund Institute of Technology, 1998.

Eri 91

Eriksson, L-H.

A Finite Version of the Calculus of Partial Inductive Definitions.

In [WELP 2].

Eri 92a

Eriksson, L-H.

Pi Users Manual

SICS 1992

Eri 92b

Lars-Henrik Eriksson

Ph. D. thesis in preparation

to be presented at the Department of Computer Science, University of Stockholm, during 1992.

Fre 90

Fredholm, D.

On function definitions I

Basic Notions and Primitive Recursive Function Definitions

Lic. Dissertation - Dept. of Computer Science, Chalmers University of Technology, Göteborg Sweden, 1990.

FS 92

Fredholm, D.; Serafimovski S.

Partial Inductive Definitions as Type-Systems for λ -terms

in Nordström, B. (ed.), *Programming Logic*, Proceedings of a workshop on

Programming Logic held in Båstad, Sweden in June 1989. Published as a Special Issue of the BIT Journal: BIT 32:1, Göteborg 1992.

GCLA 90

Aronsson, M.; Eriksson, L-H; Gäredal, A.; Hallnäs, L.; Olin, P.

The programming Language GCLA: A definitional approach to Logic Programming

In *New Generation Computing* Vol. 7, no. 4, 1990; pp. 381-404. Also available as SICS Research Report R89005B, Swedish Institute of Computer Science, Stockholm 1989.

GCLA 91

Aronsson, M.; Eriksson, L-H.; Hallnäs, L.; Kreuger, P.

A Survey of GCLA: A Definitional Approach to Logic Programming

In [WELP 1].

Gir 87

Girard J.-Y.

Linear Logic

In *Theoretical Computer Science* Vol. 50, no. 1, 1987; pp. 1-102.

Hal 86

Hallnäs, L.

On the Interpretation of Inductive Definitions

Research Report SICS R86005, Swedish Institute of Computer Science, Stockholm 1986.

Later versions published as [Hal 91a] with new title.

Hal 87a

Hallnäs, L.

A Note on Non-Monotonic Reasoning

in Brown, F. M. (ed), *The Frame Problem In Artificial Intelligence*, Proceedings of the 1987 Workshop. Morgan Kaufman 1987.

Hal 87b

Hallnäs, L.

A note on the logic of a logic program

in Dybjer, P et. al. (ed) *Proceedings of the Workshop on Programming Logic*, PMG-R 37, University of Göteborg and Chalmers University of Technology, Göteborg 1987

Hal 89

Hallnäs, L.; Nordström, B

A Definitional view of Functional Programming

in Dybjer, P et. al. (ed) *Proceedings of the Workshop on Programming Logic*, PMG-R 54, University of Göteborg and Chalmers University of Technology, Göteborg 1989

Hal 91a

Hallnäs, L.

Partial Inductive Definitions

In *Theoretical Computer Science* Vol. 87(1), 1991. Earlier versions published in: A. Avron et. Al. Editors, *Workshop on General Logic*, Report ECS-LFCS-88-52. Dept. of Computer Science, University of Edinburgh, 1987 and as SICS Research Reports R86005B and R86005C, Swedish Institute of Computer Science, Stockholm 1988.

Hal 91b

Hallnäs, L.

Models of partial inductive definitions

In Plotkin, G.; Huet, G. (eds.), *Logical Frameworks* (pp. 365-384). Cambridge University Press 1991.

Hal 92a

Hallnäs, L.

On Systems of Definitions, Induction and Recursion

in Nordström, B. (ed.), *Programming Logic*, Proceedings of a workshop on Programming Logic held in Båstad, Sweden in June 1989. Published as a Special Issue of the BIT Journal: BIT 32:1, Göteborg 1992.

Hal 92b

Hallnäs, L.

Logical and computational invariants of programs

In [WELP 2].

Han 92

Hanschke, P.

Terminological Reasoning and Partial Inductive Definitions

In [WELP 2].

HSH 90

Hallnäs, L.; Schroeder-Heister, P.

A Proof Theoretic Approach to Logic Programming

In *Journal of Logic and Computation*, vol. 1(2), 1990 and vol. 1(5), 1991. An earlier version was published as SICS Research Report R88005, Swedish Institute of Computer Science, Stockholm 1988.

JSW 86

Proceedings of the Fourth Japanese-Swedish workshop on Fifth Generation Computer Systems held at Skokloster July 1986

Swedish Institute of Computer Science (SICS), Stockholm 1986.

Kow 79a

Kowalski, R.

Algorithm = Logic + Control

CACM Number 7, July 1979. Vol. 22. pp. 424-436.

Kow 79b

Kowalski, R

Logic for Problem Solving

Elsevier Science Publishers, Amsterdam 1979.

Kre 92

Kreuger, P.

GCLA II - A definitional approach to control

SICS Research Report R92:09, Swedish Institute of Computer Science, Stockholm 1992, also published (minus some minor additions and corrections) as Lic. Dissertation - Dept. of Computer Science, Chalmers University of Technology, Göteborg Sweden, 1990 and in [WELP 2].

Ll 87

Lloyd J. W.

Foundations of Logic Programming (Second Edition)

New York 87.

MDA 90

Barklöf, K. (red)

Datorstöd för platskontoret (in Swedish)

Dokumentation från MDA-symposium, Elektrum-Kista, mars 1990

Sekretariatsrapport, 1990:5, Arbetsmiljöfonden och Styrelsen för Teknisk Uveckling i samarbete, Stockholm, Sweden, Augusti 1990.

Mil 89

Miller D.

Abstractions in Logic Programming

In Odifreddi P. (ed.), *Logic in Computer Science*, Academic Press 1989. Also published as Research report MS-CIS-89-30, LINC LAB 152, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia.

Mil 90

Miller D.

Abstractions in Logic Programming

In [WELP 1]. Also published as Research report MS-CIS-90-54, LINC LAB 182, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia.

Pal 89

Palmkvist, J.

Implementation of a Planning system in GCLA

SICS Research Technical Report T89018, Swedish Institute of Computer Science, Stockholm 1989. This work was also presented as the thesis for Palmkvists masters degree in computer science.

PS 89

Paulson, L. C.; Smith, A. W.

Logic Programing, Functional Programming and Inductive Definitions

In [WELP 1].

Rob 65

Robinson J. A. A.

Machine-Oriented Logic Based on the Resolution Principle.

Journal of the Association for Computing Machinery, Vol. 12, No. 1 January 1965.

RT 88

Brian Ritchie and Paul Taylor

The Interactive Proof Editor: An Experiment in Interactive Theorem

Report ECS-LFCS-88-61, Department of Computer Science, University of Edinburgh, 1988.

S-H 91

Schroeder-Heister, P.

Structural Frameworks, substructural logics and the role of elimination inferences.

In Plotkin, G.; Huet, G. (eds.), *Logical Frameworks* (pp. 385-403). Cambridge University Press 1991.

S-H 92

Cut-elimination in logics with definitional reflection.

In Pearce, D.; Wansing, H. (eds.), *Non-Classical Logics and Information Processing*. Springer Lecture Notes in Artificial Intelligence, Springer-Verlag (in press).

SS 86

Sterling, L.; Shapiro, E.

The Art of Prolog

MIT Press, Cambridge, Massachusetts, 1986.

WELP 1

P. Schroeder-Heister (Ed.)

Extensions of Logic Programming, International Workshop, Tübingen, FRG, Dec. 1989 Proceedings

Springer Lecture Notes in Artificial Intelligence, Vol. 475, Springer-Verlag 1991

WELP 2

Eriksson, L-H; Hallnäs, L; Schroeder-Heister, P. (Eds.)

Extensions of Logic Programming - ELP 91

Proceedings of the Second Workshop on Extensions of Logic Programming held at SICS, Stockholm, Sweden January 1991

Springer Lecture Notes in Artificial Intelligence 596, Springer-Verlag 1992.

WELP 3

Lamma, E. Mello, P (Eds.)

Currently (June 1992) only in the form of a pre proceedings:

3:rd International Workshop on Extensions of logic Programming held at Bologna February. 26-28, 1992

Facoltà di Ingegneria, Università di Bologna, Italy 1992