LiTH-IDA-R-88-15

ISSN-0281-4250

INSTITUTIONEN FÖR DATAVETENSKAP

UNIVERSITETET OCH TEKNISKA HÖGSKOLAN LINKÖPING

LINCKS: An Imperative Object Oriented System

by

Lin Padgham

Ralph Rönnquist

RESEARCH REPORT
RKLLAB, June 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

The Department of Computer and Information Science
Linköping University

PhD theses:

* Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

(*Linköping Studies in Science and Technology. Dissertations.*)

*No 77	**Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982.
No 94	**Hans Lunell:** Code Generator Writing Systems, 1983.
No 97	**Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983.
*No 109	**Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984.
*No 111	**Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984.
*No 155	**Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons. 1987.
No 165	**James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning. 1987.
No 170	**Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems. 1987.

Licentiate of engineering Theses:

(*Linköping Studies in Science and Technology. Theses.*)

No 48	**Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching. 1985.
No 52	**Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems. 1985.
No 60	**Johan Fagerström:** Simulation and Evaluation of an Architecture based on Asynchronous Processes. 1985.
No 71	**Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System. 1987.
No 72	**Tony Larsson:** On the Specification and Verification of VLSI Systems. 1986.
No 73	**Ola Strömfors:** A Structure Editor for Documents and Programs. 1986.
No 74	**Christos Levcopoulos:** New Results about the Approximation Behaviour of the Greedy Triangulation. 1986.
No 104	**Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology. 1987.
No 108	**Rober Bilos:** Incremental Scanning and Token-based Editing. 1987
No 111	**Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments. 1987.
No 113	**Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge. 1987.
No 118	**Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987
No 126	**Dan Strömberg:** Transfer and Distribution of Application Programs. 1987
No 127	**Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems. 1987
No 139	**Christer Bäckström:** Reasoning about Interdependent Actions. 1988.
No 140	**Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing. 1988.

LiTH-IDA-R-88-15 ISSN-0281-4250

LINCKS: An Imperative Object Oriented System

by

Lin Padgham

Ralph Rönnquist

Abstract: We present the LINCKS (Linköpings Intelligent Knowledge Communication System) system which is under development at Linköping University, and the theoretical model on which it is based. LINCKS is an object oriented system in the sense that objects in the real world are represented directly within the system, allowing for a clear and simple mapping from the real world to a computer model of some aspect of the real world. The object oriented model used differs significantly from that used in Smalltalk, Simula, and many other object oriented languages. The most fundamental differences are in the view of typing and the view of actions. A typing schema is regarded as part of the context for a user or a software tool, with many different typing schemas being allowed to co-exist within the system. Actions are not represented as messages and methods attached to objects as in Smalltalk. Instead actions have their own structure and organisation, and include information about their pre and post conditions. This gives the system a more imperative nature than many object oriented systems. A history mechanism is also described which allows recreation of any object as it existed at some previous point in time. The history mechanism also provides information to be used by reasoning software. LINCKS is compared to Smalltalk with respect to some important differences.

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

1. Introduction

In the 1980's, object oriented languages and object oriented environments have become increasingly popular. Probably the most well known and well developed such language is SMALLTALK [GOL83]. Some other specifically object oriented languages or environments are SIMULA [DAH70], LOOPS [BOB81], and Modula. The object oriented programming style has also been quite widely applied to conventional (i.e. non object oriented) languages [COX84]. Some examples of these are Object Oriented C, ModPascal [OLT85], Object Oriented Programming in Prolog [ZAN84], etc.

Most of the existing systems are fairly similar in their view of an object oriented system or language. As a result of this 'object-oriented' has become almost synonymous with 'SMALLTALK-like'. In this paper we will present a system LINCKS, and the underlying object oriented model for the system. This model and system have a number of important differences from the SMALLTALK environment.

We will first present the LINCKS system, which is currently just beginning the implementation phase at Linköpings University. We will then review the major differences between LINCKS and SMALLTALK, discussing the costs and the advantages of these. (We will compare only with SMALLTALK as it is the most well known of the object oriented languages, and has become something of a standard for comparisons. As far as we know the other systems are similar to SMALLTALK in the areas where we differ).

2. Description of LINCKS

The LINCKS system consists of an object oriented data base and a number of sophisticated tools and services to make this data base flexible, powerful and useful, both for individual and cooperative work.

2.1 Object Model

Objects are understood as things out-side the system, so that internally, they are only *denoted* and *described.* Denotions of objects are distinct entities used in reference to the objects in question. However, the denotions do not give any information about the objects denoted, except that two different objects are assumed to have two different denotions.

Information about objects are contained in *feature descriptors*, each of which describe some features without attachment to any particular object. That is, a feature descriptor contains information that describes some collection of features that are "things that can be known about some object".

The current knowledge about the objects is represented by associating object denotions to feature descriptors. The *database state* is therefore a mapping from the set of object denotions into the set of feature descriptors.

Each feature descriptor contains some amount of information, and there is a relation, $\sqsubseteq$, that corresponds to the notion of "more information" on feature descriptors. This relation forms a lattice,which means that the object denotions are associated to points in the information lattice.

The basic data structure used to denote and describe objects is a *molecule.* Molecules may contain three different kinds of information: an image, plain attributes and relationships to other objects. The image is a perception of the object which is

intelligible to the user, but not necessarily to the system - primarily text, graphics, and bitmaps. Attributes are a collection of name-value pairs which give a description of a number of properties of the object. Relationships between objects are represented as two way labelled links. Both attributes and links are understood and used by the system, and it is these that constitute the feature descriptor of the molecule.

Large objects are usually represented as a structure of molecules. A single molecule is the denotion for the given object, and it is linked to denotions of its parts, which then also are viewed as objects. For example, a report may be made up of a number of sections, each of which in turn is made up of a number of paragraphs.

The molecule concept supports the creative process of ordering and bringing together thoughts. Each molecule is usually a relatively small information chunk, which can be arranged and rearranged with other such information chunks. Good ideas which are initially disjoint, can be saved and later integrated into theories, papers, ideas, etc.

2.1.1 Object Typing

Object typing is assumed to reflect similarities between the descriptions of objects that are of same type. In the lattice of feature descriptors, the maximal similarity between two object descriptions d_1 and d_2 is their greatest lower bound, $d_1 \sqcap d_2$.

For any type t, norm(t) is the feature descriptor containing the information that is required for all objects that are considered to be of type t. The system knows an object denoted by g to be of type (denoted by) t if and only if the feature descriptor d currently associated to g is above the current norm(t)).

We call this view of typing *descriptive typing*, since the actual type(s) for an object is derived from what is known about the object and what is required for the type(s). The type(s) for an object is not a property of the object, but a property of the (systems) view of and knowledge about objects.

This enables several typing schemas to be used. For instance, each user in a multi-user system (like LINCKS) can have his own typing schema.

Prescriptive typing, as opposed to descriptive, is obtained by stating expicitely in the description of an object that it is of a certain type. This is important to include for the reasons that the information contained cannot be expected to be complete, and that it needs to be changed as the environment (the world) changes.

To include prescriptive typing, we cut the feature descriptor associated to an object denotion into parts, so that the information that corresponds to the type requirements are left out, and replaced by links to the denotions of the respective types. In other words, the complete feature descriptor for an object is built from the explicit information combined with the explicit information (the type norm) of its prescribed types. The picking of parts to add to this combination may continue recursively if the type norms also are typed prescriptively.

An effect of prescriptive typing is that any change to the information that describes a type t will result in changes to the information describing the objects prescribed to be of type t. The prescriptive typing model also conserves storage, since the information required for all objects of type t need to be stored explicitely only as the type norm.

The information that describes a type includes both the type *norm*, which is what information is required in the descriptions of objects of that type, and a type *seed*, which is what information is expected in the descriptions of objects of that type. The type seed can also be understood as a normal case description that is used as default when a new object is encountered and stated to be of the particular type.

In prescriptive typing, the complete feature descriptor is built using the type seed rather than the type norm, but when the seed information is in conflict with the instance level information the latter will override.

Operations on objects are defined with respect to what they require of the feature descriptors rather than being associated to types. An operation applies to (the internal representation of) an object independent of the object's type(s), provided that the information describing the object satisfies the requirement for the operation. This point is an important difference from traditional object oriented systems (e.g. Simula, SmallTalk, etc.), and it has the advantage that redefinition of typing can be done with minimal concern for the operations and vice versa.

2.2 Actions/Operations

In discussing operations on objects, it is important to be clear about the distinction between an operation *in terms of actual code* (the technical point of view), and an operation *in terms of effect* (the usage point of view), more or less the code versus the interpretation of its effect. It is clear, that the interpretation of the effect of performing an operation on one type of objects may be different from the interpretation of the effect of performing (technically) the same operation on another type of objects. Likewise, one operation performed on one type of objects may be understood as the same as or similar to another operation performed on another type of objects. In the following, we refer to the actual code as an operation, and to the interpretation of the effect of an operation as an *action*.

2.2.1 Operations

Operations on objects are defined along with constraints on the descriptors for the objects upon which it applies. Typically, these constraints are lower bounds on the information contained, which is the same kind of constraint as is ensured by prescriptive typing. The statement of constraints in terms of explicit feature descriptors, rather than as prescriptive type names, has two advantages. First, it allows the use of the same operations by users with different typing schemas. Secondly, it ensures that operations are not over constrained because they must be linked to a point where a type is defined.

Operation definitions also contain information about the effects on the feature descriptors of the objects to which they are applied. Whereas the operation constraints are pre-conditions (to be satisfied before the operation can be applied), the effect is regarded as post conditions. This information can then be used in planning what operations can be used to pave the way for other operations.

2.2.2 Actions

Operations are conceptually grouped into *actions* within particular contexts. Those operations which are understood as similar by a particular person, intelligent service, or tool such as an editor are grouped together and referred to by a single action name. Both the arrangement of the groupings, and the naming schema may differ from one context to another. When an action is requested, it is requested both in a context and with respect to actual parameters. The context's information about its naming schema, and its grouping of operations, is then used to resolve the action request into an operation (code). When checking the constraints for the different operations, the operation that is selected is the one that has constraints below and closest to the actual parameters.

A context has associated with it, information about various aspects of actions. There is information about which operations are grouped into which conceptual actions, and the naming schema for those actions. This enables a particular action request to be

translated into different operations, within differing contexts. This flexibility is analogous to the flexibility of allowing different typing schemas for different users.

Each action has associated with it, two types of information about itself. The first is an abstraction of similar pre and post conditions in the operations contained within the action. For example, a create action, which consists of different create operations, may have as abstracted pre condition 'type label X', and abstracted post condition, 'object of type X'. The second type of information is information about the meaning or interpretation of the action within the context. This is also in the form of pre and post conditions, and can be used by planning services to generate meaningful structures (partially ordered) of actions.

It should be noted that actions can contain either operations, or structures of operations and/or actions. This gives a nested model of actions, and allows for simple and complex actions.

The pre conditions for an action do not constrain the interactive use of actions by human agents, but they do guide and constrain the use of actions by *intelligent services.*

2.3 Intelligent Services

Intelligent Services is the term used for software which is able to initiate and do intermediate or high level tasks without direct user control. Many activities done by users have clearly defined goals, but no clearly defined methods. That is, there are a wide number of possible ways that the goals may be accomplished. Intelligent services also have the flexibility to find many alternative ways of achieving their goals. If one method fails for some reason, they can seek other ways of achieving their ends.

Most intelligent services need to be adapted to, or built specially for some particular situation. The specification of actions and operations along with pre and post conditions simplifies the design and implementation of intelligent services. New actions can be composed from existing actions and operations, in goal oriented ways, by general purpose planning services.

2.4 History Mechanisms

There are three different types of history information which are available in LINCKS.

Object history
: Information about how a given object looked at a particular point in time. (e.g. show me report X as it was when the last printout was made?)

Text history
: Information about the evolution of a certain instance of an object. This is often identical to the above, but at times is not. (e.g. A working paper may be incorporated into book A as chapter Y. A history of the evolution of that piece of text which is chapter Y of book A, shows the development of the working paper, as well as any development since it was incorporated into book A).

Command history
: Information about the commands that have been performed previously and their results.

The history mechanisms incorporates the notion of parallelism through partially ordered time. Two people may develop the same paper separately for a time, later merging their efforts back into a single version of the paper. The two parallel development paths are stored as such in the history.

The history information is contained in three different structures.

version structure
stores information about the development of an object over time.

parent structure
stores information about the edit history of a particular image over time.

command history structure
stores information about commands given and their effects.

The history information allows for the recreation of objects as they existed at any past point in time. It also allows for examination of the effects ofactions, and use of this information by intelligent services.

2.5 Portables

An integral part of the LINCKS system is the portable laptop computer. It can be connected to the main system, loading various portions of the data base, to allow work to be done remotely. On return from a remote working session, the new work is integrated with the main data base. It is, of course, also possible to use the portable as a connected workstation.

Portables, due to size and processing limitations, do not supply the complete functionality available on the main system. However, they do 'know about' the main system, so that tasks which cannot be done are queued to be executed later by the main system. Thus, the user has a sense of having done those tasks on the portable (in that he has given the commands and need no longer think about them), even though the execution actually occurs at a later point in time.

3. Discussion and Conclusion

The essential differences between LINCKS and SMALLTALK lie in three main areas: the *typing schema*, the *action representations* and *action schema*, and the *history mechanisms*. There are also some additions in the LINCKS system (i.e. portables and intelligent services) which could theoretically be added to a SMALLTALK system, without any restructuring of the basic system. We will discuss the essential differences, but not the additions.

3.1 Typing Schema

The major difference between the LINCKS view of typing and that of SMALLTALK, results from two things: the descriptive typing of LINCKS, where types are seen as similarity in feature descriptors, and the fact that operations are NOT defined strictly on types.

The fact that the LINCKS system has an objective way of comparing objects feature descriptors (and thus type feature descriptors also), means that different users can have totally different type structures. There is no requirement for these to be made compatible on the basis of naming or of organisation. The system can facilitate communication between users with different languages (typing schemas), on the basis of its own objective knowledge.

For example, user A has a *document type*, whereas user B has a *text object type* and types for several sorts of documents such as a *report type*, a *note type*, etc. The system knows (and can communicate to B) that when A mentions something of type *document* to B, that in B's terms that is a thing of type *text object*, though a little

more specialised.

LINCKS also allows multiple inheritance within the prescriptive typing schema. This is not allowed in SMALLTALK, and sometimes causes problems in mapping to the real world. There are ways to get around this limitation in SMALLTALK using *classes* (ZDO?), or some similar construct.

3.2 Action Representation and Schema

In SMALLTALK, actions are somewhat secondary to objects. They are represented as procedures attached to types. All invocation of actions result from message passing between objects. In LINCKS, actions and operations are structured and organised on the same level as objects.

There is no connection in LINCKS between objects and operations, except for those which result from execution constraints. (E.g. If one has an operation to change the value of a header attribute, then the object operated on needs to contain a header attribute). These differences from the SMALLTALK approach have both costs and benefits.

The cost of separating objects and operations is that you no longer have the same level of data abstraction, where a part of an objects definition consists of what you can do with it. Also one loses the strict control over what can be done with objects that results from allowing only those operations which are strictly defined for the object. It can be argued that this leads to a less robust and more error prone system.

The benefits of the LINCKS approach are however a system which we hope will prove more flexible and more powerful. The removal of operations from the type definitions makes the definition of type schemas much more accessible to the end user. In SMALLTALK, it is the programmer or application builder who sets up the type hierarchy for the end user. In LINCKS, the end user sets up and defines his own type schema. We consider that organisation and naming of types is very much a matter of personal opinion. Consequently, it is appropriate for it to be a part of the user interface rather than an integral part of the system.

The representation of actions as things which can be organised and structured in their own right, gives LINCKS a much more action oriented approach. It has a more imperative and goal oriented nature than SMALLTALK. The augmenation of actions and operations with pre and post conditions facilitates the building of expert systems. These are seen as existing with contexts and being a part of the overall system. The notion of contexts, and separate communicating agents (human or software) within the system, is much more developed in LINCKS than in SMALLTALK.

3.3 History Mechanisms

LINCKS has extensive and thorough history mechanisms, which are totally absent in SMALLTALK. The lack of such mechanisms in SMALLTALK is a serious drawback for at least some applications. There are some attempts to build such a mechanism onto SMALLTALK [], but it is not clear how difficult or successful these attempts are.

The obvious possible drawback of such history mechanisms is the cost in terms of time and memory. Just how great this cost is can be better assessed after implementation, but we are hopeful that response time can be kept reasonable, at least for those things which are done commonly and often.

The history mechanisms provide a great deal of contextual and developmental information which can be used to enable the system to perform more 'intelligently'. (E.g. The information is available to respond to a request such as 'retrieve the text

that was section 6 in the most recent printout of report X').

3.4 Conclusion

The breaking away in LINCKS from the SMALLTALK tradition of having operations as methods defined on objects, is quite controversial. However, we believe there are solid grounds for this breakaway, both on the basis of a theoretical model, and on the grounds of desirable functionality. We consider that LINCKS maintains the essential and desirable quality of an object oriented system, where objects in the real world are represented directly as those objects within the system. This facilitates simple and clear mapping to the external world. Whether LINCKS will in practice turn out to be as popular and useful as SMALLTALK reamins to be seen. However, we consider the ideas to be both novel and sound, and therefore well worth a try.

4. *Bibliography*

1. Aiello, L., Nardi, D. and Panti, M. *Structural Office Modelling: A first step towards the office expert system.* Proceedings of 2nd ACM Conference on Office Information Systems, June 25-27 1984, Toronto, Canada, p. 161-169

2. Barbi, E., Calvo, F., Perale, C., Sirovich, F., Turini, F. *A Conceptual approach to Document Retrieval.* 1984 ACM

3. Biggerstaff, Ted J., Endres, D. Mack, Forman, Ira R. *Table: Object Oriented Editing of Complex Structures.* 1984 IEEE 334-345

4. Bobrow, D.G., Stefik, M. *The LOOPS Manual,* Tech. Rep. KB-VLSI-81-13, Knowledge Systems Area, Xerox Palo Alto Research Centre.

5. Bracchi, Giampio Pernici, Barbara *The Design Requirements of Office Systems.* ACM Transactions on Office Information Systems, Vol 2, No 2, April 1984, p. 151-170.

6. Cox, B.J. *Message/Object Programming: an evolutionary change in programming technology.* IEEE Software, vol. 1, no. 1, Jan. 1984 p.50-61.

7. Croft, Bruce W. Lefkowitz, Lawrence S. *Task Support in an Office System.* ACM Transactions on Office Information Systems, 1984 Vol. 2, Issue 3, p. 197-212.

8. Dahl, O-J, Myhrhaug, B., Nygaard, K. *Simula Common Base Language* Norsk Regnescentral, Oslo 1970.

9. Goldberg, A., Kay, A. *SMALLTALK-80: The Language and Its Implementation,* Addison-Wesley, 1983.

10. Kimura, Gary D. Shaw, Alan C. *The Structure of Abstract Document Types.* Second ACM Conference on Office Information Systems, June 25-27 1984, Toronto, Canada, p. 161-169

11. Lin, Wente K., Ries, Daniel R., Blaustein, Barbara T. Chilenskas, R. Mark *Office Procedures as a Distributed Database Application.* Data Base, Winter 1984, 5-10

12. Lochovsky, F.H. A Knowledge Based Approach to Supporting Office Work. Database Engineering, IEEE Computer Society, Vol. 16, No. 3, Sept. 1983, p. 43-51

13. Olthoff, W. *An Overview on ModPascal* SIGPLAN Not. vol. 20, no. 10 Oct 1985, p. 60-71.

14. Sandewall, E. *Self-description and reproduction in distributed programming systems.* Informatics Laboratory, Linköping University, report number LITH-MAT-R-79-22, August 1979.

15. Tichy, W.F. *Design, Implementation and Evaluation of a Revision Control System*, Proceedings of the 6th International Conference on Software Engineering, 1982

16. Ullman, J.D. *Principles of Database Systems*, Computer Science Press Inc., USA, 1982.

17. Zanilo, C. *Object-oriented Programming in Prolog.* 1984 International Symposium on Logic programming, Atlantic City, NJ, USA 6-9 feb. 1984 p.265-709

18. Zdonik, Stanley B. *Object Management System Concepts.* SIGOA Newsletter, Second ACM-SIGOA Conference on Office Information Systems, vol. 5, no. 1-2, p.13-19.

19. Zisman, M. *Use of Production Systems for Modelling Asynchronous, Concurrent Processes.* In Pattern Directed Inference Systems, Waterman and Hayes-Roth, Eds., Academic Press, New York, 1978, 53-68

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-15

Upplagans storlek:
Number of copies:

Datum:
Date: June 1988

Projekt:
Project:

Titel:
Title: LINCKS: An Imperative Object Oriented System

Författare:
Author: Lin Padgham, Ralph Rönnquist

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

Abstract: We present the LINCKS (Linköpings Intelligent Knowledge Communication System) system which is under development at Linköping University, and the theoretical model on which it is based. LINCKS is an object oriented system in the sense that objects in the real world are represented directly within the system, allowing for a clear and simple mapping from the real world to a computer model of some aspect of the real world. The object oriented model used differs significantly from that used in Smalltalk, Simula, and many other object oriented languages. The most fundamental differences are in the view of typing and the view of actions. A typing schema is regarded as part of the context for a user or a software tool, with many different typing schemas being allowed to co-exist within the system. Actions are not represented as messages and methods attached to objects as in Smalltalk. Instead actions have their own structure and organisation, and include information about their pre and post conditions. This gives the system a more imperative nature than many object oriented systems. A history mechanism is also described which allows recreation of any object as it existed at some previous point in time. The history mechanism also provides information to be used by reasoning software. LINCKS is compared to Smalltalk with respect to some important differences.

Nyckelord (högst 8):
Keywords (8):

Bibliotekets anteckningar

A Selection of Previous Research Reports.

LiTH-IDA-R-88-06 **Christer Bäckström:** A Representation of Coordinated Actions Characterized by Interval Valued Conditions.

LiTH-IDA-R-88-05 **Christer Bäckström:** Keeping and Forcing: How to Represent Cooperating Actions.

LiTH-IDA-R-88-04 **Wlodek Drabent, Simin Nadjm-Tehrani, Jan Maluszynski:** Algorithmic Debugging with Assertions.

LiTH-IDA-R-88-03 **Staffan Bonnier, Jan Maluszynski:** Towards a Clean Amalgamation of Logic Programs with External Procedures. Accepted to the *Joint Fifth Int. Logic Programming Conf. and Fifth Logic Programming Symp.*, Seattle Washington, August 15-19, 1988.

LiTH-IDA-R-88-02 **Ulf Nilsson:** Inferring Restricted AND-Parallelism in Logic Programs using Abstract Interpretation.

LiTH-IDA-R-88-01 **Jacek Leszczylowski, Staffan Bonnier, Jan Maluszynski:** Logic Programming with External Procedures: Introducing S-Unification. A shorter version is to appear in *Information Processing Letters*, North Holland.

LiTH-IDA-R-87-26 **Jonas Löwgren:** Applying a Rapid Prototyping System to Control Panel Dialgues.

LiTH-IDA-R-87-24 **Sven Moen:** Drawing Dynamic Trees.

LiTH-IDA-R-87-22 **Lars Ahrenberg, Arne Jönsson:** An Interactive System for Tagging Dialogues. Also in *Proc. of 14th Conference of the Association for Literary and Linguistic Computing*, Göteborg 1-5 June, 1987.

LiTH-IDA-R-87-21 **Harold W. Lawson, Jr.:** Challenges and Directions in Computers and Education.

LiTH-IDA-R-87-20 **Krzysztof Kuchcinski, Zebo Peng:** Parallelism Extraction from Sequential Programs for VLSI Applications. This paper is to appear in *Microprocessing and Microprogramming, the Euromicro Journal*, 1988.

LiTH-IDA-R-87-19 **Zebo Peng:** Construction of Asynchronous Concurrent Systems from their Behavioral Specifications. Also in *Proc. of the 10th World Computer Congress*, Dublin, Ireland, Sept. 1-5, 1986.

LiTH-IDA-R-87-18 **Henrik Nordin:** Reuse and Maintenance Techniques in Knowledge-Based Systems.

LiTH-IDA-R-87-17 **Tony Larsson:** Specification and Verification of VLSI Systems Actional Behaviour This is a close version of a paper presented at *the 8th international conference on Computer Hardware Description Languages, CHDL,* 87.

LiTH-IDA-R-87-16 **Shamsul I. Chowdhury:** State of the Art in Statistical Expert Systems. Also in *Proc. of the Conference on Expert Systems and their Applications*, Avignon, May 1987.

LiTH-IDA-R-87-15 **Nils Dahlbäck:** Kommunikation med datorer i naturligt språk - vad är det och vem behöver det?

LiTH-IDA-R-87-14 **Peter Fritzson:** Window System Architectures - an Overview. Also presented at *A Tutorial on Software Development Environments,* Linköping University, Nov 24-25 1986.

LiTH-IDA-R-87-13 **Mats Wirén:** A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing. Also in *Proc. of the Third Conference of the European Chapter of the Association for Computational Linguistics,* Copenhagen, April 1—3, 1987.

LiTH-IDA-R-87-11 **Sture Hägglund, Christer Hansson, Tomas Sokolnicki:** Knowledge-Based Training of Case Management Routines and Emergency Procedures. Also in *Proc. of the 3rd Int. Conf. on Expert Systems*, London, 1987.

LiTH-IDA-R-87-10 **Andrzej Lingas:** On Parallel Complexity of the Subgraph Isomorphism Problem.

LiTH-IDA-R-87-09 **Andrzej Lingas, Marek Karpinski:** Subtree Isomorphism and Bipartite Perfect Matching are Mutually NC Reducible.

The Department of Computer and Information Science
Linköping University

organizes undergraduate and graduate studies in *Computer Science*, *Telecommunication and Computer Systems*, and *Administrative Data Processing*. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- *ACTLAB - Laboratory for Complexity of Algorithms*, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.
- *AIELAB - Artificial Intelligence Environments Laboratory,* which conducts research on representation and manipulation of knowledge, problem solving techniques and expert systems with mechanical engineering applications.
- *ASLAB - Application Systems Laboratory,* which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.
- *CADLAB - Laboratory for Computer-Aided Design of Electronics*, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.
- *LIBLAB - Laboratory for Library and Information Science,* which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.
- *LOGPRO - Laboratory for Logic Programming*, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.
- *NLPLAB - Natural Language Processing Laboratory*, which conducts research related to the development and use of natural language interfaces to computer software.
- *PELAB - Programming Environments Laboratory*, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.
- *RKLLAB - Laboratory for Representation of Knowledge in Logic*, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

Research Reports 1988

LiTH-IDA-R-88-15	**Lin Padgham, Ralph Rönnquist:** LINCKS: An Imperative Object Oriented System. Also in *Proc. of the 20th Annual Hawaii Int. Conf. on System Sciences*, Hawaii, 1987.
LiTH-IDA-R-88-13	**Erik Tengvald:** Ett kartorienterat planeringssystem för autonoma farkoster, en design diskussion.
LiTH-IDA-R-88-12	**Erik Sandewall, Johan Hultman, Erik Tengvald:** Software Architecture and Programming Paradigms for Robotics Applications.
LiTH-IDA-R-88-11	**Mats Wirén:** An Incremental Chart Parser for PATR.
LiTH-IDA-R-88-10	**Mats Wirén:** A Control-Strategy-Independent Parser for PATR. Revised version of paper appearing in *Proc. of First Scandinavian Conf. on Artificial Intelligence*, Tromsö, Norway, March 9 - 11, 1988.
LiTH-IDA-R-88-09	**F** [illegible] g.
LiTH-IDA-R-88-08	**E** [illegible] ing about Change with R [illegible]
LiTH-IDA-R-88-07	**E** [illegible] 87.