A bird's-eye view of LISPLOG:

The LISP/PROLOG integration with initial-cut tools

Harold Boley (Editor)

December 1987 SWP-86-08
Third Edition

A bird´s-eye view of LISPLOG:

The LISP/PROLOG integration with initial-cut tools

Harold Boley (Editor)

Abstract:

A combined LISP/PROLOG system was designed, implemented and tested, via stepwise refinement of Kenneth M. Kahn´s operational LISP semantics for pure PROLOG.

LISPLOG.1 utilizes the representation of PROLOG terms as LISP-S-expressions for two generalizations of Edinburgh PROLOG : varying length structures, and goals with predicate variables. On the other hand, a **goto>while**-like spezialization is studied in this language: the **cut>initial cut** restriction, which improves both readability and parallelization of PROLOG programs.

For accessing LISP from PROLOG, we permit LISP predicates as goals, and LISP functions as right-hand sides of the is-predicate.
In the other direction, the first n PROLOG solutions can be returned as a LISP list.

LISPLOG.2 augments the trace and break tools already available in LISPLOG.1 by an (initial) ´cut-indicator´/´manual cutter´ for making the cuts in the search tree observable and interactive.

For improving efficiency, the originally recursive interpreter is reformulated iteratively, the binding environment is represented as an array structure, and the database is indexed by predicates and arguments.

As the main application, LISPLOG runs a knowledge-based system, μ-UNIXPERT, for diagnosing printing problems.

This research is much obliged to the Land Rheinland-Pfalz and the Deutsche Forschungsgemeinschaft, SFB 314

General information about LISPLOG

Mail address:

LISPLOG-Projekt, Raum 14/403 *(after 1 March 1988:* Raum 12/403*)*
Fachbereich Informatik
Universitaet Kaiserslautern
Postfach 3049

D-6750 Kaiserslautern
W.-Germany

E-MAIL address:

uucp: unido!uklirb!lisplog

- or -

lisplog@uklirb.UUCP

Telephone:

LISPLOG office : (0631) 205 -2805
Secretary : -2802

Current Structure of the LISPLOG Project:

Consultation
Harold Boley

Performance
Iterative Interpreter
Michael Dahmen
Clause indexing
Ansgar Bernardi

Concepts
Modules and Streams
Michael Dahmen
Comparison with other Approaches
Franz Kammermeier

Interaction
Zoom-Box Model and Initial-Cut Tools
Manfred Meyer

Application
Bridge
Michael Dahmen
Minichess
Manfred Meyer
Stability of Differential Equations
Juergen Herr
micro-UNIXPERT
Michael Lessel

Translators
LISPLOG to CPROLOG
Knut Hinkelmann
CPROLOG to LISPLOG
Michael Dahmen

Portation
Michael Dahmen
Franz Kammermeier
Ansgar Bernardi

Contents:

1. How to get the LISPLOG system and documentation

The LISP source of the LISPLOG system is available both in FRANZ LISP for UNIX™ systems and in COMMON LISP.

We deliver the program via E-MAIL (UUCP network) or on tape (format tar or standard ANSI with 800 or 1600 bpi).
Currently we charge no copy fee.

The printed documentation (cf. chapter 4) is sent by ordinary mail, free of charge.
You can speed up orders by enclosing your mail address on a self-adhesive label. Remaining delays may be due to reprinting.

™ UNIX is a registered trade mark of Bell Laboratories

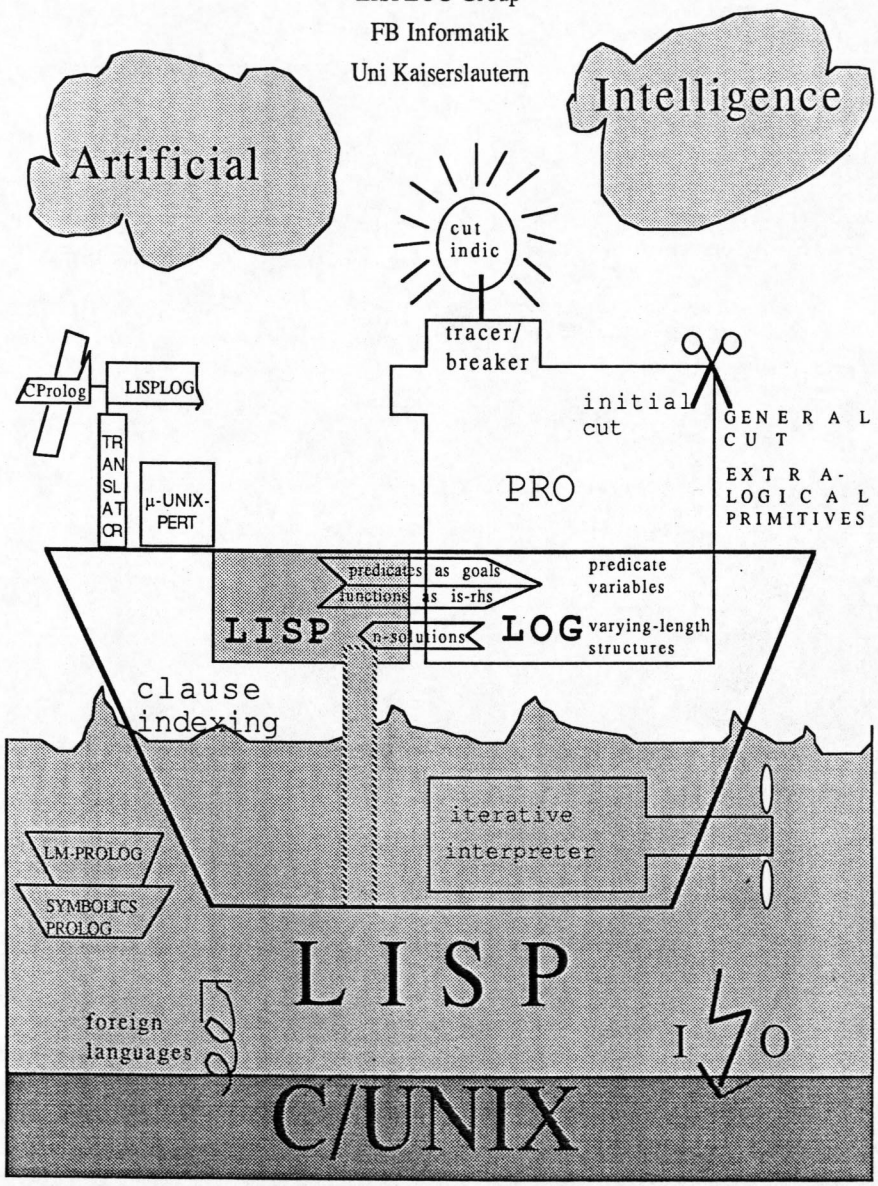
2. LISPLOG: A LISP-based LISP/PROLOG integration

This chapter is a collection of overhead transparencies, versions of which have been used for oral presentations of the LISPLOG project since 6 November 1986.

LISPLOG

A LISP-based LISP/PROLOG Integration

LISPLOG Group
FB Informatik
Uni Kaiserslautern



General motivation

- Study a PROLOG model at a LISP workbench

  Examine PROLOG-related AI(XPS) **formalisms** ("shells") above instruction level.
  Develop PROLOG-in-LISP **merger** under AI application control.
  Prepare logical/functional **synthesis** beyond PROLOG/LISP.

- Bridge gap between Horn-logic theory and XPS practice

  Augment Horn clauses by function-call premises.
  Permit certain higher-order features, such as **predicate variables.**
  Introduce types,modules,data structures,etc.
  Supply options for search control more readable than general **cuts.**
  Resort to LISP for lower-level algorithms.

- Gain experience in LISP/PROLOG-based AI languages

  Acquire project-specific know-how about the entire spectrum of
  - Conceptual design
  - Operational semantics
  - Efficient implementation
  - Interactive interface
  - AI application

  Apply this know-how to
  - Assess available languages more accurately.
  - Tailor them to one's own requirements.
  - Develop new tools or languages if/when necessary.
  - Provide the accumulating software to others.

Design&Implementation Approach: Stepwise Refinement of Kahn's Interpreter

Embed Kahn's PROLOG into FRANZ LISP (LISPLOG.0).

Then modify it (LISPLOG.1):

Generalization :	Structures of varying lengths (e.g.: appimp).
	Goals with predicate variables (e.g.:someq/everyq).
Specialization :	Cut operator only as **initial** premise (e.g.:fac).
	Edinburgh primitives excised if too extra-logical.

And integrate them (LISPLOG.1):

LISP-from-PROLOG access :
Call LISP predicates as ground goals.
Call LISP functions as right-hand sides of **is** primitive.

PROLOG-from-LISP access :
Return first n solutions as a LISP list.

Add interactive support :

Tracer/Breaker	(LISPLOG.1)
Cut tools	(LISPLOG.II+2)

Finally improve efficiency :

Recursion removal	(LISPLOG.2)
Environment handling	(LISPLOG.2)
Clause indexing	(LISPLOG.II+2)
Edinburgh translator	(LISPLOG.1+2)

Long-term R&D Goal: Provide LISP Alternative to Edinburgh PROLOG

- Higher **purity** implies easier :
  Understandability (e.g.:acquisition of new team members)
  Portability (e.g.:transfer to SYMBOLICS COMMON LISP)

- User-friendly **interface** (see above)

- Acceptable **efficiency** (see above)

- Testbed for **innovative concepts** in functional/logical languages :

Lazyness	(streams)
Goal reducer	(functional pre-reduction of logical goals à la LOGLISP)
Cut options	(initial&dynamic cuts, stream cuts)
Modules	(pragmatic concept, integrated with clause indexing)

-**Application** case studies :

Bridge & Chess	
LLshLL.0	(Frame & Production Shell)
μ-UNIXPERT	(UNIX printer diagnosis)
SPADE	(difference-methods stability analysis)

Sample LISPLOG Programs

Varying-length structures: **appimp**

```
(ass (appimp nil))
(ass (appimp _res _firstarg . _remarg)
     (append-r _firstarg _int _res)
     (appimp  _int . _remarg))
(ass (append-r nil _list _list))
(ass (append-r (_first . _rest1) _list (_first . _rest2))
     (append-r _rest1 _list _rest2)
```

Predicate variables: **someq** and **everyq**

```
(ass (someq _pred _arg . _remarg) (_pred . _arg))
(ass (someq _pred ID . _remarg) (someq _pred . _remarg))

(ass (everyq _pred))
(ass (everyq _pred _arg . _remarg)
     (pred . _arg)
     (everq _pred . _remarg))


(ass (knows john _x))
```

Initial cut: **fac**

```
(ass !(fac 0 1))
(ass (fac _x _res)
     (is _z (sub1 _x))
     (fac _z _z1)
     (is _res (times _x _z1)))
```

The LISPLOG Interactive Programming Environment

Manfred A. Meyer
Universität Kaiserslautern

General motivation:

- The development of functional/relational programs with LISPLOG requires an integrated and interactive programming environment which should be both powerful and easy to use.

- It should consist of:
  - an integrated tracer supporting backtracing for LISP functions and PROLOG relations
  - a break-package with powerful debugging aids activated via breakpoints or interrupts
  - a step-by-step execution mode
  - special innovative tools supporting the non-standard concepts of LISPLOG like the initial-cut operator

- There should be no overhead caused by the programming environment when running programs without using the debugging tools.

How to trace a LISPLOG program ?

(1) Use LISP trace to watch the execution of the LISPLOG program:

advantage: usefull for tracing the functional part of the LISPLOG program
disadvantage: no information about the execution of LISPLOG predicates

(2) Use the standard box model (by L. Byrd) instead:

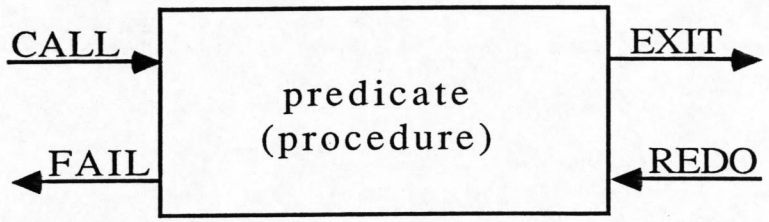
advantage: good model for describing the operational semantics of PROLOG
disadvantage: no representation for the functional part of LISPLOG

(3) Extend the box model in order to represent the evaluation of LISP functions too:

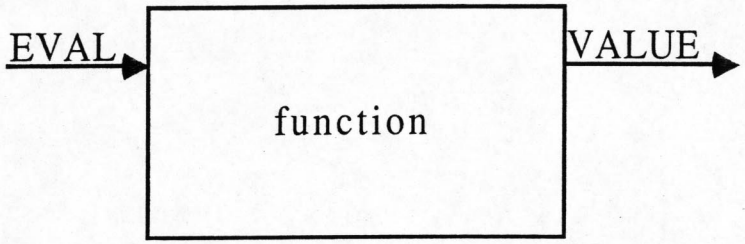
advantage: well suited (integrated) model for describing both the functional and the relational part of the LISPLOG program

The extended box model for LISPLOG:

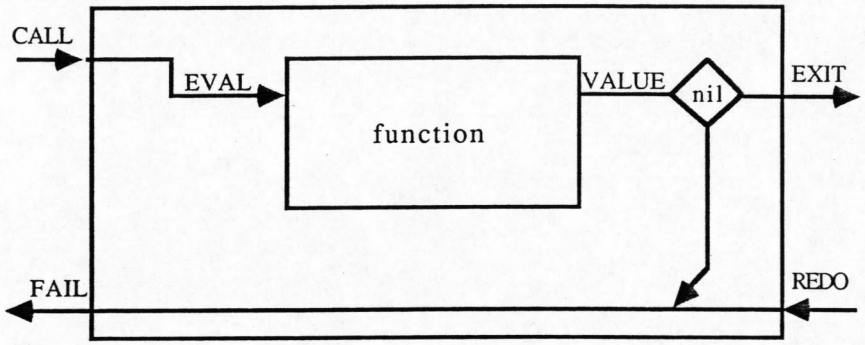
- box model for predicates:



- box model for functions:



- box model for function (LISP) calls as goals:



What is the interesting information about execution of LISPLOG programs?

- control-flow through the box ports CALL, EXIT, FAIL, REDO, EVAL, and VALU(E):

Example:

Tell LISPLOG about natural numbers:

```
(ass (natural-number 1))
(ass (natural-number _n) (natural-number _m)
                         (is _n (add1 _m)))
```

And how square numbers are defined:

```
(ass (square _x) (natural-number _n)
                 (is _x (times _n _n)))
```

Ask LISPLOG: *"Is it true that 4 is a square number?"*

```
*(square 4)
|CALL (square 4)
| CALL (natural-number _n-1)
| EXIT (natural-number 1)
| CALL (is 4 (times 1 1))
|  EVAL (times 1 1)
|  VALU 1
| FAIL (is 4 (times 1 1))
| REDO (natural-number _n-1)
|  CALL (natural-number _m-2)
|  EXIT (natural-number 1)
|  CALL (is _n-1 (add1 1))
|   EVAL (add1 1)
|   VALU 2
|  EXIT (is 2 (add1 1))
| EXIT (natural-number 2)
| CALL (is 4 (times 2 2))
|  EVAL (times 2 2)
|  VALU 4
| EXIT (is 4 (times 2 2))
|EXIT (square 4)
success!
```

- results of unification:
  - success: new variable-bindings = Δ unifier = θ_{new} - θ_{old}
  - failure: "mismatch"
  - backtracking: withdrawn variable-bindings = last Δ unifier

Example:

Define the append relation for lists:

```
(ass (append nil _list _list))
(ass (append (_first . _rest1) _list (_first . _rest2))
     (append _rest1 _list _rest2))
```

And then try to prove the goal: *(append (1 5) _x (1 2 3 4))*

```
*(append (1 5) _x (1 2 3 4))
|CALL (append (1 5) _x (1 2 3 4))
| unify: {_first-1/1, _rest1-1/(5), _list-1/_x,
|          _rest2-1/(2 3 4)}
| CALL (append (5) _list-1 (2 3 4))
|  mismatch!
| FAIL (append (5) _list-1 (2 3 4))
| backtrack: {_first-1/1, _rest1-1/(5), _list-1/_x,
|             _rest2-1/(2 3 4)}
|FAIL (append (1 5) _x (1 2 3 4))
nil
```

- module switches:

Example: (taken from the minichess program):

```
|CALL (ckeck-situation (F W W W W F F F F F F B B B B F))
| CALL (not (bad-situation (F W W W W F F F F F F B B B B F)))
|  switch-to-module minichess-memory
|  CALL (bad-situation (F W W W W F F F F F F B B B B F))
|  FAIL (bad-situation (F W W W W F F F F F F B B B B F))
|  switch-to-module minichess-main
| EXIT (not (bad-situation (F W W W W F F F F F F B B B B F)))
|EXIT (ckeck-situation (F W W W W F F F F F F B B B B F))
```


- initial-cut indicator:
  prints information about the number of clauses cut and how many of them are applicable:

Example:

Define the faculty function as PROLOG relation:

```
(ass !(fac 0 1))
(ass (fac _n _nf) (is _n1 (sub1 _n))
                  (fac _n1 _n1f)
                  (is _nf (times _n _n1f)))
```

```
*(fac 1 _x)
|CALL (fac 1 _x)
| CALL (is _n1-1 (sub1 1))
|  EVAL (sub1 1)
|  VALU 0
| EXIT (is 0 (sub1 1))
| CALL (fac 0 _n1f-1)
| EXIT (fac 0 1)
| CALL (is _nf (times 1 1))
|  EVAL (times 1 1)
|  VALU 1
| EXIT (is 1 (times 1 1))
|EXIT (fac 1 1)
success!
(_x = 1)
```

Call for more solutions:

```
*more
|REDO (fac 1 1)
| REDO (fac 0 1)
|  Cutting 1 [1] clauses after clause (ass !(fac 0 1))
| FAIL (fac 0 _n1f-1)
|FAIL (fac 1 _x)
nil
```

All information is stored in a backtrace stack.
This guarantees full backtrace-possibility while paging through the backtrace stack!

The LISPLOG break-package:

The LISPLOG break-level is a special read-eval-print-loop accepting all commands available at the LISPLOG toplevel.

Furthermore, there are some special commands available only at the LISPLOG break-level giving the user some helpful information about the current state of the LISPLOG program:

- INFO
- ENVIRONMENT
- GOAL
- PATH
- LIST-OF-GOALS
- CONTINUE
- RESET
- SKIPPING
- EVAL <s-expr>

New commands for the LISPLOG toplevel:

commands for activating the box model tracer:

- spy {<box-name>}*
- nospy {<box-name>}*

commands for setting and removing breakpoints:

- brk {<box-name> | (<box-name> {<port>}*)}*
- nobrk {<box-name> | (<box-name> {<port>}*)}*

Moreover, the LISPLOG break-level is not only activated when reaching breakpoints, but may also be entered via asynchronous interrupts (by CTRL-C) or directly from the manual cutter.

commands for controlling the manual cutter:

- cut {<pred-name>}*
- nocut {<pred-name>}*

commands for skipping internal information about the execution of some boxes (zooming):

- skip {<box-name>}*
- skipq {<box-name>}*
- noskip {<box-name>}*

Innovative cut tools:

- cut indicator: is part of the extended tracer.

- manual cutter:

  If backtracking has to be done for some goal
  and the manual cutter is activated for this goal
  and there is another applicable clause for it
  then first ask the user if he really wants the system to try the alternative clause:

  "Do you want to cut the clause (ass (fac 0 1))?"

  (At this time the user may enter the LISPLOG break-level in order to get more detailed information about the current state of the execution before he decides whether to let backtracking take place or not.)

  If the user denies, LISPLOG will act as usual.
  Otherwise, the system acts as if the failed clause has been temporary marked with the initial cut operator. Furthermore, it asks:

  "Do you want to make the cut permanent?"

  If the user agrees, the cut is inserted in the database in order to avoid backtracking for this clause permanently.

Summary:

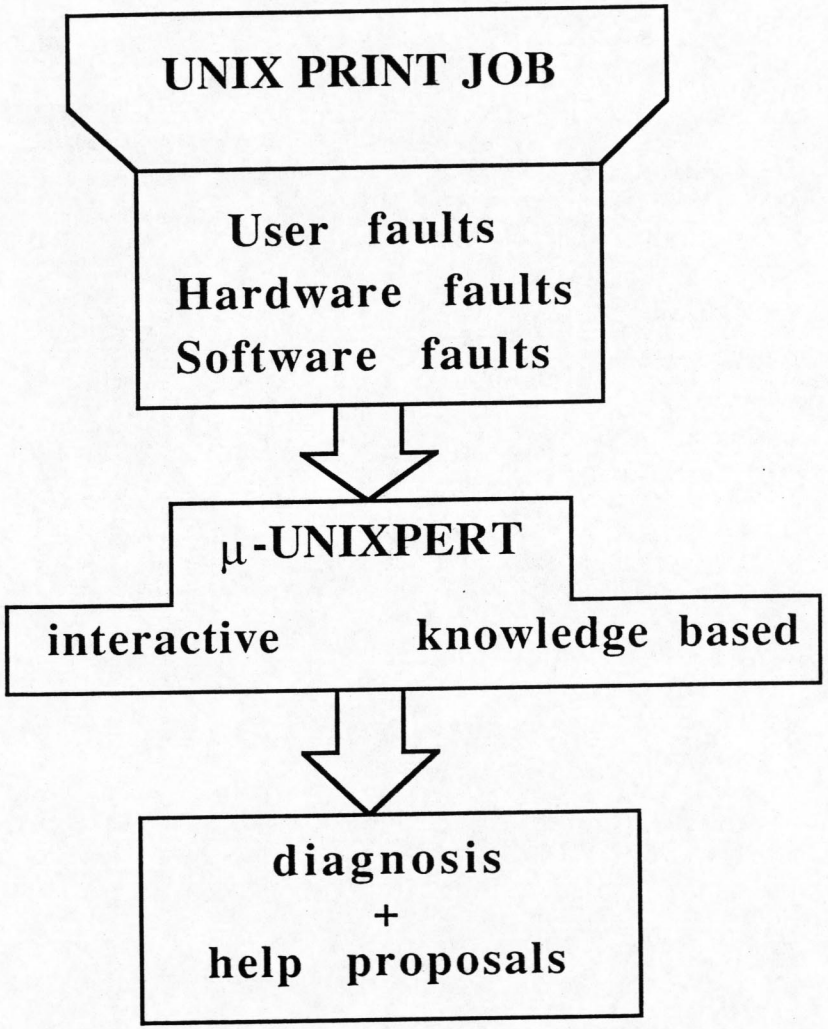
- The LISPLOG Programming Environment makes the execution of functional/logical programs visible to the user, and supports debugging LISPLOG programs by several tools.

- It consists of
  - an integrated tracer based on an extended zoom box model for LISP function calls and PROLOG goals,
  - a comfortable break-package activated via breakpoints or user interrupt (CTRL-C),
  - some innovative tools to support programming with the initial cut operator (manual cutter, cut indicator), and
  - a step-by-step execution mode.

- It is
  - optional (code automatically loaded when needed),
  - modular (seperate trace- and break-package),
  - universal (well-defined interface; same code for LISPLOG.2 and LISPLOG.II),
  - transparent (implemented almost without using impure LISP constructs), and
  - integrated in the standard LISPLOG system without any overhead (space or time) in normal execution mode.

μ-UNIXPERT

(UNIX-PERiphery Tester)
Michael Lessel



Why LISPLOG ?

Properties of μ-UNIXPERT:

-Information Acquisition

-Access to LISP functions
-dynamic Expansion
of the Knowledge Base

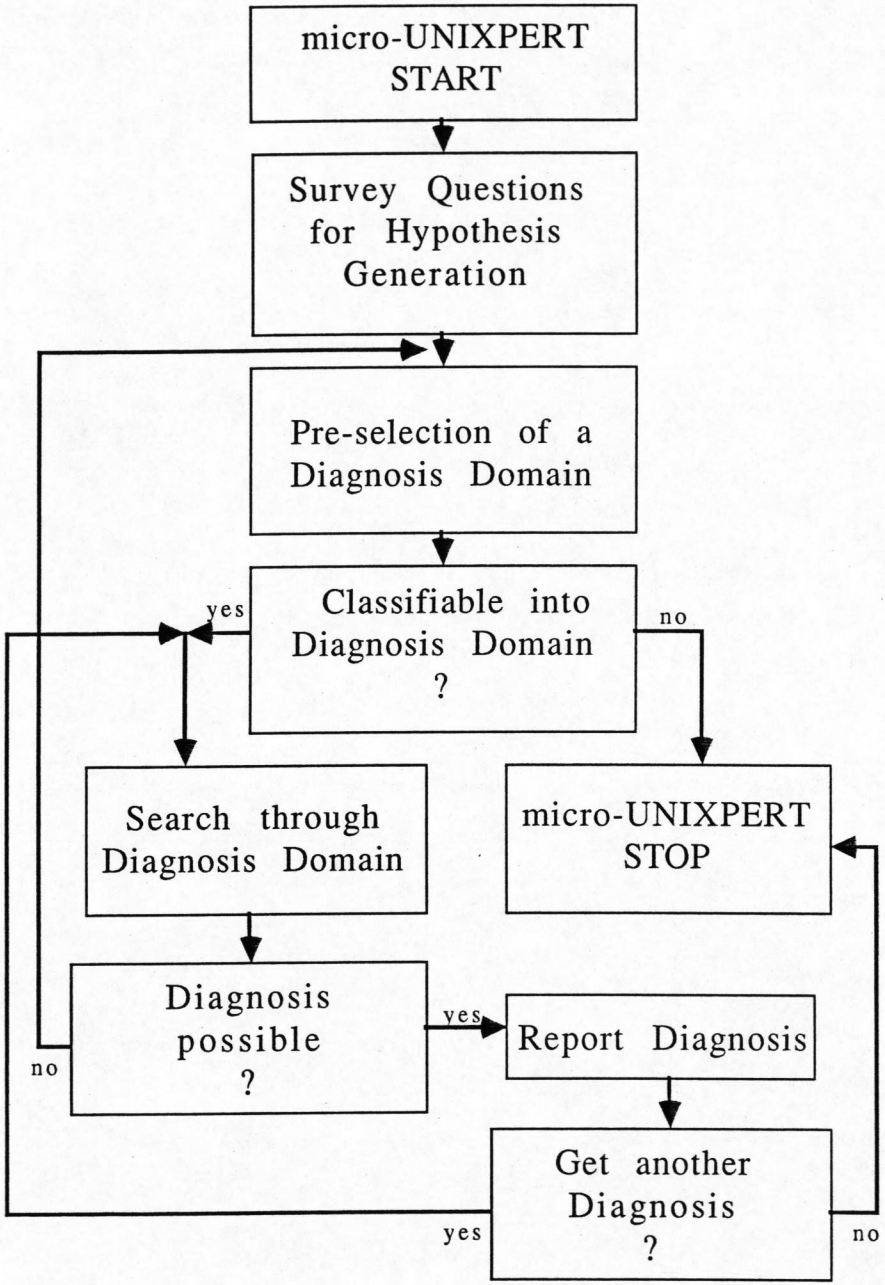
-Capability of Self-Explanation

-Logic Deduction

-Depth-first search in a Search Tree

-Incremental expansion

⇒ LISPLOG supports all points above



LISPLOG and the Other Ones

Franz Kammermeier

e.g.: **Symbolics Prolog**
LM-Prolog
Babylon-Prolog

Symbolics Prolog:

- commercial product
- high performance implementation
- specific for Symbolics LISP Machines

LM-Prolog:

- initially experimental workbench, enhanced to a commercial product
- flexible and transparent implementation, considerably optimized (compiler, microcode)
- specific for LMI LISP Machines

Babylon-Prolog:

- (only) one component of a hybrid system
- object oriented implementation, non-optimized
- specific for Symbolics + TI LISP Machines

LISPLOG:

- experimental workbench
- flexible and transparent implementation, modestly optimized
- machine independent (FRANZ LISP, UNIX) portable (e.g. Common LISP)

Feature comparison

Innovative concepts

Symbolics Prolog	+/-
LM-Prolog	++
Babylon-Prolog	+/-
LISPLOG	+

Programming Environment

Symbolics Prolog	+/++
LM-Prolog	++
Babylon-Prolog	+/++
LISPLOG	+

Modularization

Symbolics Prolog	+
LM-Prolog	+
Babylon-Prolog	-
LISPLOG	+

Feature comparison (continued)

Efficiency

Symbolics Prolog	++	(≈50 kLIPS) compiled
LM-Prolog	+	(≈6 kLIPS) compiled
Babylon-Prolog	-	(≈0,1 kLIPS) interpreted
LISPLOG	+/-	(≈0,3 kLIPS) interpreted

Flexibility

Symbolics Prolog	--
LM-Prolog	+
Babylon-Prolog	+
LISPLOG	+

Portability

Symbolics Prolog	--
LM-Prolog	-
Babylon-Prolog	-
LISPLOG	+

Innovative features of LISPLOG

Restriction to initial cut

to achieve more transparency

New interactive features

Initial-cut tools
(cut indicator and manual cutter)
to control the cut
Extended box-model tracer
to integrate predicate/function
debugging

Autoquoter for LISP expressions

which tries to distinguish code and data

LISPLOG-CPROLOG Translator

which accepts combined
functional/relational programs

Generator returning PROLOG solutions

to make use of backtracking from LISP

Iterative Interpreter for LISPLOG

Michael Dahmen

LISPLOG.1

-**recursive** and **purely functional**

-**all** information stored in lists

-**structure copying**

LISPLOG.2

-**iterative** control structure

-use of **arrays**

-partial **structure sharing**

-**improved unification**

Main goal

optimizing of runtime and memory requirements

Performance of LISPLOG.1

length of prove : limited by LISP stack

speed : 20 LIPS (VAX 750, FRANZ LISP)

Performance of LISPLOG.2

length of prove : unlimited

speed : 100 LIPS (VAX 750, FRANZ LISP)
300 LIPS (SYMBOLICS, COMMON LISP)

Module system for LISPLOG

MODULE = part of LISPLOG database

controlled interaction between modules

similar concept as in

COMMON LISP

ADA

MODULA-2

Example:

Bridge card-playing program

4 players

view of **each player** represented in **one module**

another module contains **common knowledge**

Indexing the LISPLOG Database

Ansgar Bernardi

Problem:

Improve Efficiency of LISPLOG

Method:

Elimination of Unnecessary Unifications

by

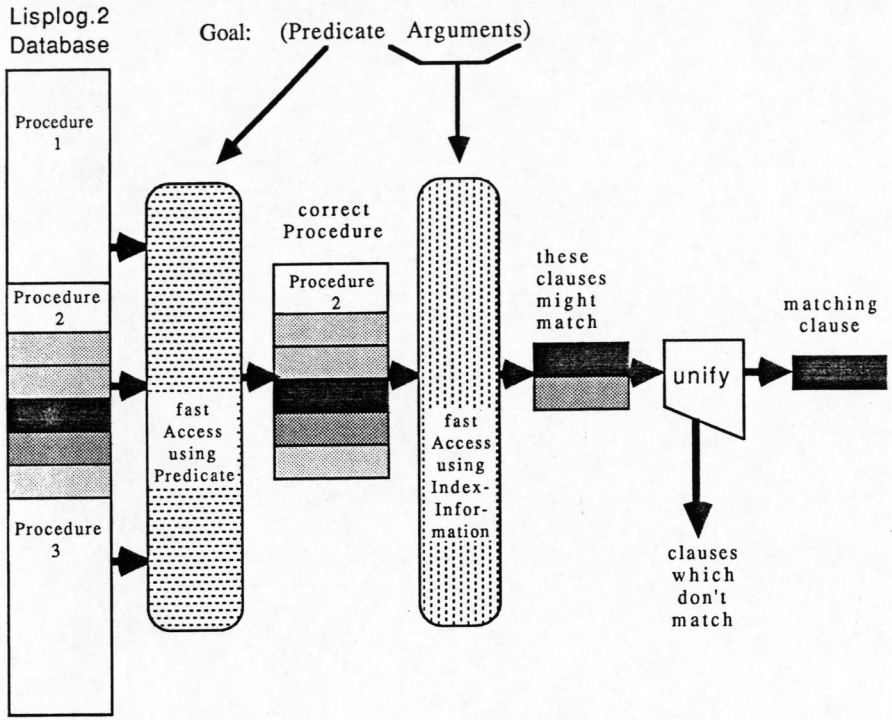
Indexing of Clauses

using

its Predicate

and

One of its Constant, Atomic Arguments



Concentration on Fast Retrieval

⇒ while **asserting/retracting** a **clause**

- **save the order**
- **special treatment** of
  variables
  complex terms
  variable arity

Use of Indexing depending on

- **goals** (variables ?)
- **structure of the database**
  (# clauses / procedure)

Possible influence

- indexed **argument selectable** for **every procedure**

- **indexing** may be **disabled** for **any procedure**

of

- **user**
- **program**

at any time

LISPLOG-CPROLOG Translator

Knut Hinkelmann

CPROLOG - Program:

Set of Clauses

LISPLOG - Program:

Set of Clauses
+
Set of Function Definitions

Proof strategy :

CPROLOG :

Resolution

LISPLOG :

Resolution + Evaluation

Phase 1: Clause Translation

(nested)	**function calls**
(conjunction of)	**relation calls**

Because of applications of the LISP functions COND,OR,AND one LISPLOG clause may be translated to several CPROLOG clauses.

Phase 2: Transformation of function definitions to CPROLOG clauses

3 kinds of functions :

- predicates
- nonpredicative functions
- pseudo functions

Examples :

1. premise :

```
(member  a (append _l1 _l2))
->
append(_l1,_l2,_r08),
member(a,_r08).
```

2. premise :

```
(is _x (length (member a _l)))
->
member(a,_l,_r09),
length(_r09,_x).
```

3. premise :

```
(progn    (print _x)
          (princ ":")
          (writeblanks (diff 20 (flatsize1 _x)))
          t)
->
write(_x),
write( : ),
flatsize1(_x,_r12),
_r11 is (20 - _r12),
writeblanks$p(_r11),
true.
```


Example :

```
(def member
   (lambda (a 1)
         (cond  ((null l) nil)
                ((eq a (car l)) l)
                (t (member a (cdr l))))))
```

----->
predicate

```
member(_a,[]) :-
        fail.

member(_r26,[_r26|_x29]).

member(_a,[_r27|_x30]) :-
        not(_a == _r27),
        member(_a,_x30).
```

--------->
non-predicate

```
member(_a,[],[]).

member(_r18,[_r18|_x22],[_r18|_x22]).

member(_a,[_r19|_x23],_r21) :-
        not(_a == _r19),
        member(_a,_x23,_r21).
```

Example :

```
clause:  (ass (test _x _y)
              (cond  ((greaterp 3 _x) (eq a _y))
                     ((lessp 3 _x) (eq b _y))
                     (t (eq c _y))))

--->    test(_x,_y) :-
                3 > _x,
                a == _y.
        test(_x,_y) :-
                not(3 > _x),
                3 < _x,
                b == _y.
        test(_x,_y) :-
                not(3 > _x),
                not(3 < _x),
                true,
                c == _y.
```

with Cut:

```
---->   test(_x,_y) :-
                3 > _x,
                !,
                a == _y.
        test(_x,_y) :-
                3 < _x,
                !,
                b == _y.
        test(_x,_y) :-
                true,
                !,
                c == _y.
```


3. A LISPLOG sample dialog

```
Script started on Wed Nov 12 15:14:35 1986
1:Lisplog

Franz Lisp, Opus 38.89 mit Cmu-, Ktu- und Flavors-Erweiterungen
08.02.1985

Patch Nr. 38.3 geladen
Patch Nr. 38.4 geladen
Patch Nr. 38.5 geladen
LISPLOG System geladen.

Diese Version umfasst :
  - LISPLOG Interpreter Version 2
  - Box Modell Tracer
  - Modul System
  - schaltbare Quotierungsautomatik
  - Streams
  - Indexierungssystem

starte LISPLOG mit  (lisplog)
weitere Informationen dann mit   help

Auf dem LISPLOG-Toplevel habe alle Kommandos die Form :
  <Kommando> {<argumente>}*

Namen, die keine Kommandos sind,
werden als Anfragen eines LISPLOG-Goals interpretiert.

Die Form (<g1> {<a1>}*) (<g2> {<a2}*) ...  kann benutzt werden,
um mehrere konjunktiv verknuepfte Ziele zu beweisen.


News sind mit der Funktion (news) zu erhalten   Stand : 00-00-0000
1.[user] lisplog
*[user] destroy
t
*[user] consult ../datenbasen/appimp.db
[load ../datenbasen/appimp.db]
nil
*[user] l

(ass (appimp nil))
(ass (appimp _res _firstarg . _remarg)
```

```
    (append-r _firstarg _int _res)
    (appimp _int . _remarg))

(ass (append-r nil _list _list))
(ass (append-r (_first . _rest1) _list (_first . _rest2))
    (append-r _rest1 _list _rest2))


t
*[user] appimp _x (a b) (c d) (e) (f)
success :
((_x = (a b c d e f)))
*[user] more
nil
*[user] (appimp (a b c d e f) _x _y _z)
success :
((_z = (a b c d e f)) (_y = nil) (_x = nil))
*[user] more
success :
((_z = (b c d e f)) (_y = (a)) (_x = nil))
*[user] more
success :
((_z = (c d e f)) (_y = (a b)) (_x = nil))
*[user] more
success :
((_z = (d e f)) (_y = (a b c)) (_x = nil))
*[user] more
success :
((_z = (e f)) (_y = (a b c d)) (_x = nil))
*[user] more
success :
((_z = (f)) (_y = (a b c d e)) (_x = nil))
*[user] more
success :
((_z = nil) (_y = (a b c d e f)) (_x = nil))
*[user] more
success :
((_z = (b c d e f)) (_y = nil) (_x = (a)))
*[user] more
success :
((_z = (c d e f)) (_y = (b)) (_x = (a)))
*[user] (appimp _x (a b) _y (c d))
success :
((_y = nil) (_x = (a b c d)))
*[user] more
success :
```



```
((_y = (_first-6)) (_x = (a b _first-6 c d)))
*[user] more
success :
((_y = (_first-6 _first-7)) (_x = (a b _first-6 _first-7 c d)))
*[user] more
success :
((_y = (_first-6 _first-7 _first-8))
 (_x = (a b _first-6 _first-7 _first-8 c d)))
*[user] more
success :
((_y = (_first-6 _first-7 _first-8 _first-9))
 (_x = (a b _first-6 _first-7 _first-8 _first-9 c d)))
*[user] more
success :
((_y = (_first-6 _first-7 _first-8 _first-9 _first-10))
 (_x = (a b _first-6 _first-7 _first-8 _first-9 _first-10 c d)))
*[user] more
success :
((_y = (_first-6 _first-7 _first-8 _first-9 _first-10 _first-11))
 (_x = (a b _first-6 _first-7 _first-8 _first-9 _first-10 _first-11 c d)))
*[user] abolish appimp
t
*[user] l

(ass (append-r nil _list _list))
(ass (append-r (_first . _rest1) _list (_first . _rest2))
    (append-r _rest1 _list _rest2))


t
*[user] destroy
t
*[user] + (fac 0 1)
t
*[user] + (fac _x _y) (is _z (- _x 1)) (fac _z _z1) (is _y (* _x _z1))
t
*[user] l

(ass (fac 0 1))
(ass (fac _x _y) (is _z (- _x 1)) (fac _z _z1) (is _y (* _x _z1)))


t
*[user] (fac 0 _x)
success :
((_x = 1))
```

```
*[user] (fac 1 _x)
success :
((_x = 1))
*[user] (fac 3 _x)
success :
((_x = 6))
*[user] spy fac
(fac)
*[user] (fac 3 _x)
|CALL (fac 3 _x)
| CALL (fac 2 _z1-1)
|  CALL (fac 1 _z1-2)
|   CALL (fac 0 _z1-3)
|   EXIT (fac 0 1)
|  EXIT (fac 1 1)
| EXIT (fac 2 2)
|EXIT (fac 3 6)
success :
((_x = 6))
*[user] more
|REDO (fac 3 _x)
| REDO (fac 2 _z1-1)
|  REDO (fac 1 _z1-2)
|   REDO (fac 0 _z1-3)
|    CALL (fac -1 _z1-4)
|     CALL (fac -2 _z1-5)
|      CALL (fac -3 _z1-6)
|       CALL (fac -4 _z1-7)
|        CALL (fac -5 _z1-8)
|         CALL (fac -6 _z1-9)
|          CALL (fac -7 _z1-10)
|           CALL (fac -8 _z1-11)
|            CALL (fac -9 _z1-12)
|             CALL (fac -10 _z1-13)
|              CALL (fac -11 _z1-14)
|               CALL (fac -12 _z1-15)
|                CALL (fac -13 _z1-16)
|                 CALL (fac -14 _z1-17)
|                  CALL (fac -15 _z1-18)
|                   CALL (fac -16 _z1-19)
|                    CALL (fac -17 _z1-20)
|                     CALL (fac -18 _z1-21)
[user enters interrupt key]

<<< LISPLOG - Break - Modus >>>
Please enter Break-command: [user] r
```



```
user-break:reset-to-LISPLOG-Top-Level
*[user] cut fac
(fac)
*[user] (fac 3 _x)
|CALL (fac 3 _x)
| CALL (fac 2 _z1-1)
|  CALL (fac 1 _z1-2)
|   CALL (fac 0 _z1-3)
|   EXIT (fac 0 1)
|  EXIT (fac 1 1)
| EXIT (fac 2 2)
|EXIT (fac 3 6)
success :
((_x = 6))
*[user] more
|REDO (fac 3 _x)
| REDO (fac 2 _z1-1)
|  REDO (fac 1 _z1-2)
|   REDO (fac 0 _z1-3)
Do you want to cut clause (ass (fac 0 1))? [user] y
|  FAIL (fac 1 _z1-2)
| FAIL (fac 2 _z1-1)
|FAIL (fac 3 _x)
nil
*[user] l

(ass (fac 0 1))
(ass (fac _x _y) (is _z (- _x 1)) (fac _z _z1) (is _y (* _x _z1)))


t
*[user] - (fac 0 1)
t
*[user] assa (cut (fac 0 1))
t
*[user] spy fac
(fac)
*[user] l

(ass !(fac 0 1))
(ass (fac _x _y) (is _z (- _x 1)) (fac _z _z1) (is _y (* _x _z1)))


t
*[user] (fac 3 _x)
```

```
|CALL (fac 3 _x)
| CALL (fac 2 _z1-1)
|  CALL (fac 1 _z1-2)
|   CALL (fac 0 _z1-3)
|    Cutting 1 [1] clauses after clause (ass (cut (fac 0 1)))
|   EXIT (fac 0 1)
|  EXIT (fac 1 1)
| EXIT (fac 2 2)
|EXIT (fac 3 6)
success :
((_x = 6))
*[user] more
|REDO (fac 3 _x)
| REDO (fac 2 _z1-1)
|  REDO (fac 1 _z1-2)
|   REDO (fac 0 _z1-3)
|   FAIL (fac 0 _z1-3)
|  FAIL (fac 1 _z1-2)
| FAIL (fac 2 _z1-1)
|FAIL (fac 3 _x)
nil
*[user] l

(ass !(fac 0 1))
(ass (fac _x _y) (is _z (- _x 1)) (fac _z _z1) (is _y (* _x _z1)))


t
*[user] destroy
t
*[user] + (fac 0 1)
t
*[user] + (fac _x _y) (greaterp _x 0) (is _z (- _x 1)) (fac _z _z1) (is _y (* _x
_z1))
t
*[user] l

(ass (fac 0 1))
(ass (fac _x _y)
    (greaterp _x 0)
    (is _z (- _x 1))
    (fac _z _z1)
    (is _y (* _x _z1)))


t
```



```
*[user] spy fac
(fac)
*[user] (fac 3 _x)
|CALL (fac 3 _x)
| CALL (fac 2 _z1-1)
|  CALL (fac 1 _z1-2)
|   CALL (fac 0 _z1-3)
|   EXIT (fac 0 1)
|  EXIT (fac 1 1)
| EXIT (fac 2 2)
|EXIT (fac 3 6)
success :
((_x = 6))
*[user] more
|REDO (fac 3 _x)
| REDO (fac 2 _z1-1)
|  REDO (fac 1 _z1-2)
|   REDO (fac 0 _z1-3)
|   FAIL (fac 0 _z1-3)
|  FAIL (fac 1 _z1-2)
| FAIL (fac 2 _z1-1)
|FAIL (fac 3 _x)
nil
*[user] spy is greaterp
(is greaterp fac)
*[user] (fac 3 _x)
|CALL (fac 3 _x)
| CALL (greaterp 3 0)
| EXIT (greaterp 3 0)
| CALL (is _z-1 (- 3 1))
| EXIT (is 2 (- 3 1))
| CALL (fac 2 _z1-1)
|  CALL (greaterp 2 0)
|  EXIT (greaterp 2 0)
|  CALL (is _z-2 (- 2 1))
|  EXIT (is 1 (- 2 1))
|  CALL (fac 1 _z1-2)
|   CALL (greaterp 1 0)
|   EXIT (greaterp 1 0)
|   CALL (is _z-3 (- 1 1))
|   EXIT (is 0 (- 1 1))
|   CALL (fac 0 _z1-3)
|   EXIT (fac 0 1)
|   CALL (is _z1-2 (* 1 1))
|   EXIT (is 1 (* 1 1))
| EXIT (fac 1 1)
```

```
| CALL (is _z1-1 (* 2 1))
| EXIT (is 2 (* 2 1))
More information desired? Enter +p, -p or number of steps: [user] p
| EXIT (fac 2 2)
| CALL (is _x (* 3 2))
| EXIT (is 6 (* 3 2))
|EXIT (fac 3 6)
success :
((_x = 6))
*[user] more
|REDO (fac 3 _x)
| REDO (fac 2 _z1-1)
| REDO (fac 1 _z1-2)
| REDO (fac 0 _z1-3)
| CALL (greaterp 0 0)
| FAIL (greaterp 0 0)
| FAIL (fac 0 _z1-3)
| FAIL (fac 1 _z1-2)
| FAIL (fac 2 _z1-1)
|FAIL (fac 3 _x)
nil
*[user] translate-db nil
[fasl /usr/users/hinkelma/translator.o]
```

Translation of LISPLOG clauses

```
fac(0,1):- !.

fac(_x,_y) :-
        _x > 0,
        _z is (_x - 1),
        fac(_z,_z1),
        _y is (_x * _z1).
```

Translation of LISP functions

```
quote(X,X).
null([]).
success
*[user] destroy
t
*[user] consult /usr/users/bernardi/lisplog/timetest
[load /usr/users/bernardi/lisplog/timetest]
nil
*[user] consult index.demo.db
[load index.demo.db]
```



```
nil
*[user] l

(ass (father v1 k1))
(ass (father v1 k2))
(ass (father v1 k3))
(ass (father v1 k4))
(ass (father v1 k5))
(ass (father v1 k6))
(ass (father v2 k7))
(ass (father v2 k8))
(ass (father v2 k9))
(ass (father v2 k10))
(ass (father v2 k11))
(ass (father v2 k12))
(ass (father v3 k13))
(ass (father v3 k14))
(ass (father v3 k15))
(ass (father v4 k16))
(ass (father v4 k17))
(ass (father v4 k18))
(ass (father g1 v1))
(ass (father g1 v2))
(ass (father g2 v3))
(ass (father g2 v4))

(ass (grandparent _x _y) (father _x _z) (father _z _y))

(ass (frage) (grandparent g1 k8))

(ass (test) (time-on) (frage) (frage) (frage) (time-off) (time-print))

(ass (dotest) (test) (test) (test))


t
*[user] (dotest)
14  netto 14
15  netto 15
15  netto 15
success :
nil
*[user] index 1 father
Re-indexed
*[user] l father
(index 1 father)
```

```
(ass (father v1 k1))
(ass (father v1 k2))
(ass (father v1 k3))
(ass (father v1 k4))
(ass (father v1 k5))
(ass (father v1 k6))
(ass (father v2 k7))
(ass (father v2 k8))
(ass (father v2 k9))
(ass (father v2 k10))
(ass (father v2 k11))
(ass (father v2 k12))
(ass (father v3 k13))
(ass (father v3 k14))
(ass (father v3 k15))
(ass (father v4 k16))
(ass (father v4 k17))
(ass (father v4 k18))
(ass (father g1 v1))
(ass (father g1 v2))
(ass (father g2 v3))
(ass (father g2 v4))

t
*[user] (dotest)
11  netto 11
10  netto 10
10  netto 10
success :
nil
*[user] index 2 father
Re-indexed
*[user] l father
(index 2 father)
(ass (father v1 k1))
(ass (father v1 k2))
(ass (father v1 k3))
(ass (father v1 k4))
(ass (father v1 k5))
(ass (father v1 k6))
(ass (father v2 k7))
(ass (father v2 k8))
(ass (father v2 k9))
(ass (father v2 k10))
(ass (father v2 k11))
(ass (father v2 k12))
```



```
(ass (father v3 k13))
(ass (father v3 k14))
(ass (father v3 k15))
(ass (father v4 k16))
(ass (father v4 k17))
(ass (father v4 k18))
(ass (father g1 v1))
(ass (father g1 v2))
(ass (father g2 v3))
(ass (father g2 v4))

t
*[user] (dotest)
15  netto 15
13  netto 13
14  netto 14
success :
nil
*[user] end-demo
nil
*exit

 Auf Wiedersehen !!!

script done on Wed Nov 12 15:16:44 1986
```

4. Abstracts of the LISPLOG papers

This chapter collects the abstracts of all major LISPLOG papers written to date. For ordering information see chapter 1. Since most of the papers are in German, chapter 6 provides some help for translation.

Using the system

LISPLOG Benutzerhandbuch
Ansgar Bernardi, Michael Dahmen, Manfred Meyer

SEKI-Working-Paper 87-01

Januar 1987

ABSTRACT

This paper is intended to serve as a user's manual for LISPLOG.

It gives a short description of the syntax of LISPLOG together with an explanation of all commands available on the top-level of LISPLOG. The concepts and the commands of the LISPLOG tracer/breaker are also described.

This manual should give you the ability to use the LISPLOG system. It doesn't explain the concepts of the LISP/PROLOG integration or its implementation details.

Approaching the integration of LISP and PROLOG

L I S P L O G

MOMENTAUFNAHMEN EINER LISP/PROLOG-VEREINHEITLICHUNG

Harold Boley, Franz Kammermeier und die LISPLOG-Gruppe

MEMO SEKI-85-03

August 1985

ABSTRACT

In April 1985, the LISPLOG group began with the integration of an abstract PROLOG machine into FRANZ LISP.

This effort, here called "LISP(/PRO)LOG", is based on a very concise interpreter for PURE PROLOG in PURE LISP (Kenneth M. Kahn). First, the interpreter was slightly improved and implemented in FRANZ LISP (LISPLOG.0).

Second, the following extensions were completed by July 1985 (LISPLOG.1):

- (Nested) LISP predicates can be used like PROLOG goals; the first n PROLOG solutions can be returned to LISP.

- Selected PROLOG primitives are made available (e.g. a not primitive and a generalized is operator permitting general LISP expressions).

- An "initial" cut operator is usable for clause-choice confirmation (specializing the ordinary cut).

- Clauses are indexed according to their predicate name.

- The user interface is improved considerably (e.g. by a box-model tracer and a break-package).

Furthermore there now exists a translator for transforming a subset of LISPLOG to CPROLOG.

Currently, three applications are running in LISPLOG.1:

- A library of list-processing and set-processing relations.
- The kernel of a self-documentation system for LISPLOG.
- A program playing a chess endgame.

Two innovative interactive tools for non-deterministic languages to be explored in LISPLOG are a "cut indicator" and a "manual cutter".

LISPLOG

BEITRAEGE ZUR LISP/PROLOG-VEREINHEITLICHUNG

Michael Dahmen
Juergen Herr
Knut Hinkelmann
Harry Morgenstern

Memo SEKI-85-10

November 1985

ABSTRACT

This paper includes three sections, discussing in detail extensions and supplementary tools for the LISPLOG system.

The first contribution by Michael Dahmen discusses a translator from CPROLOG to LISPLOG, which effects a partial transformation of PROLOG-programs. This transformation is mainly based upon pattern matching, comparing structures of the program to be transformed with given patterns and substituting structures of the target language for them.
This translator is written in PROLOG, which is suited very well for this task because of its built-in capability for unification. Therefore this translator is an example of a task, which is simply and naturally solvable in PROLOG.

The second contribution by Knut Hinkelmann and Harry Morgenstern discusses the translation of programs written in LISPLOG to standard PROLOG.
In order to translate a LISPLOG program into pure PROLOG, it is necessary to flatten nested LISP function calls into a conjunction of relation calls and to translate the corresponding LISP functions into PROLOG clauses. The flattening of a nested LISP function call is treated in the first part.
The second part treats the translation of a LISP function into a conjunction of PROLOG clauses using the syntax of LISPLOG. The LISP functions are translated into pure LISPLOG.

These two contributions describe tools which enable developers of PROLOG programs to vary between the implementations of CPROLOG and LISPLOG without too much effort.

The third contribution by Juergen Herr treats the compilation of LISPLOG clauses into LISP functions and breadth-first search as an alternative control structure for LISPLOG. The first part discusses advantages and disadvantages of breadth-first search and introduces an implementation.
The discussion of possible constructions in clause heads and the development of concepts for translating LISPLOG clauses are the main themes in the second part. Furthermore an implementation of some of these concepts is described in the form of a hornclause compiler written in LISP, which assembles predefined function patterns.

LISPLOG im Kontext anderer LISP/PROLOG-Vereinheitlichungen

Franz Kammermeier

SEKI-Working-Paper 86-09

Dezember 1986

ABSTRACT

Since 1979 a lot of functional/relational languages integrating LISP and PROLOG have been described in the literature.
In this paper it is tried to find the position of LISPLOG in the context of ten important other hybrids, which are mainly LISP based, e.g. LOGLISP, LM-Prolog and Symbolics Prolog.

After a short introduction and a general view of these hybrids, firstly the integration of programming styles is compared, namely the correspondence of LISP/PROLOG data structures and the facilities to access one formalism from the other.
Implementation techniques of PROLOG interpreters, like implementation of control, representation of term instances and database indexing, are examined in the second part, with the emphasis being on LISP as the implementation language.
The comparison is supplemented by a short description of compilation techniques for PROLOG (in LISP) used by compilers of the treated languages.

Finally, some conclusions for present and future improvements of LISPLOG are outlined.

Adding interactive tools

Entwurf und Implementierung einer Interaktionsumgebung fuer LISPLOG
- 2. erweiterte Auflage -

Manfred A. Meyer

SEKI-Working-Paper 87-02

Dezember 1987

ABSTRACT

This paper deals with the design and implementation of an interactive programming environment for an integration of LISP and PROLOG called LISPLOG, which is currently developed at the Universitaet Kaiserslautern.

This programming environment consists of an extended box model tracer with backtrace possibility and a comfortable break-package.

Two innovative interactive tools for non-deterministic languages that have been explored in LISPLOG are a "cut indicator" and a "manual cutter".

In addition, an extension of the box model for the functional part of LISPLOG is discussed and prototypically implemented.

Finally, several improvements of the user interface are roughly outlined, and the current implementation is critically reviewed once again.

Improving efficiency

ITERATIVER LISPLOG INTERPRETER

Implementierung, Dokumentation und Evaluation

Michael Dahmen

SEKI-Working Paper 86-03

Juni 1986

ABSTRACT

This paper discusses the iterative interpreter LISPLOG.2, which was developed from the earlier recursive version LISP-LOG.1. The goal of this development was to speed up the execution of LISPLOG programs. A comparison between the two LISPLOG versions - given in chapter three - shows to what extent this goal could be achieved.

The first chapter of this paper describes the differences between the two versions and shows how these differences effect the performance.

Chapter two is the documentation of the main aspects of LISPLOG.2. Since LISPLOG.2 was developed from LISPLOG.1 it may be useful to consult the documentation of LISPLOG.1 too.

The appendices include the listing of the LISPLOG.2 interpreter and the programs used for the performance analysis.

Ein Indexierungskonzept für LISPLOG-Datenbasen

Ansgar Bernardi

SEKI-Working-Paper 86-10

Dezember 1986

ABSTRACT

LISPLOG is a LISP/PROLOG integration explored at the Universitaet Kaiserslautern.
To improve the performance of the LISPLOG interpreter, an indexing concept has been developed and implemented.
Higher efficiency is gained by eliminating unnecessary unifications.
Matching clauses are selected using the predicate and one constant atomic (normally the first) argument of the conclusion.
A special combination of shared lists and binary trees is maintained by the indexing procedure to make this selection rather fast.
The user may optionally specify the indexing argument; the default argument is changeable when installing the system.

The indexing concept and the implemented algorithms and data structures are fully described in this paper, together with a complete source listing (FRANZ LISP) of the implementation.

Ein Ansatz fuer einen LISPLOG Compiler mit LISP als Zielsprache

Juergen Herr

SEKI-Working-Paper 86-06

Dezember 1986

ABSTRACT

This paper describes my work on compilation of LISPLOG clauses into LISP functions and an improved runtime system for LISPLOG, in which these functions will be interpreted. The first part of this paper introduces and comments on the implemented functions, while the reasons of fundamental design decisions will be shown in the second part. Thus the first part is a program documentation, whereas the second part gives more general informations.

Furthermore the paper discusses in detail the question wether it makes sense to use (portable) LISP as the target language for a Hornclause compiler. This is the main point of the second part of this paper, and I hope it will become apparent by comparison with Warren`s abstract machine, that (portable) LISP is not very suitable for this purpose.

I have to thank Dr.L.Wiesbaum and A.Poehlmann from the Siemens corporation for papers and discussions which helped me to work on PLM (now WAM) and the descriptions of ECRC-PROLOG.

ÜBERSETZUNG VON LISPLOG-PROGRAMMEN NACH CPROLOG
- 2. erweiterte Auflage -

Knut Hinkelmann

SEKI-Working-Paper 86-05

September 1986

ABSTRACT

A LISPLOG program is a set of horn clauses. Pure PROLOG's proof strategy is a kind of resolution. In LISPLOG, however, LISP predicates can be used like PROLOG goals. These will be proved by evaluating the LISP functions instead of using the resolution theorem prover. To translate a LISPLOG program into CPROLOG, these function applications have to be transformed into applications of PROLOG relations. In order to prove these relations by resolution, new horn clauses have to be added to the database. These new horn clauses are created from the definition of the corresponding LISP function.

Extending the language

Fone and Fall: Forward-with-Backward Chaining in LISPLOG

Harold Boley

SEKI-Working-Paper 87-03
Juni 1987

ABSTRACT

A small extension for incorporating forward chaining into and on top of LISPLOG's backward chaining framework is presented.

Goal: Backward-with-Forward Chaining in LISPLOG

Harold Boley

SEKI-Working-Paper 87-04
Juni 1987

ABSTRACT

A tiny extension for performing forward chaining to prove goals set up by LISPLOG's backward chaining framework is introduced.

Frame and Heir: Clausal Frames and Multiple Inheritance in LISPLOG

Harold Boley

SEKI-Working-Paper 87-09
November 1987

ABSTRACT

Two related extensions of LISPLOG are given. The first is a frame-to-clause translator permitting 'variable-length slots' and 'goal attachment', where the clauses generated from a frame can be freely mixed with other clauses. The second is an ako-slot interpreter proving a goal by recursively inheriting information from its object's superobjects, for multiple ako links employing LISPLOG's built-in depth-first search.

Module und Streams in LISPLOG

Michael Dahmen

SEKI-Working-Paper 87-06

Juni 1987

ABSTRACT

This paper discusses two extensions of the LISPLOG system, called modules and streams.

The module system enables the LISPLOG user to build his program from a number of seperate databases (modules). The interaction between the modules is defined by special interfaces. The result is a hierarchical structure of the LISPLOG database.

The second part describes an extension of the LISP-LISPLOG interface for delayed computation of the solution of a LISPLOG inquiry. Therefore it becomes possible to compute any number of solutions without specifying that number before the computation starts. Even an infinite number of solutions may be intensional represented by means of a stream.

Diagnosing printer problems

micro-UNIXPERT: EIN WISSENSBASIERTES SYSTEM ZUR BEHANDLUNG VON PROBLEMEN BEI UNIX-DRUCKAUFTRAEGEN

Michael Lessel

SEKI-Working-Paper 86-04

Mai 1986

ABSTRACT

micro-UNIXPERT (UNIX^ - PERiphery - Tester) is an interactive, knowledge-based diagnosis system for the treatment of printer hardware faults and of other problems which can appear in connection with printing (user errors and software faults).
If possible, the system both generates diagnoses and gives proposals for clearing. The entire knowledge of micro-UNIXPERT consists of a set of PROLOG-like facts and rules.
The system is implemented in LISPLOG, a LISP/PROLOG integration running in FRANZ LISP under the UNIX 4.2 BSD operating system on a VAX 11/750 .
After a general introduction into the problems of diagnosis systems and a discussion of the most important micro-UNIXPERT features (knowledge acquisition, positive and negative facts, user acceptance, inference mechanism), the method of operation of the system is explained by giving full details of the LISPLOG implementation.

---

^ UNIX is a registered trade-mark of Bell Laboratories

μ-UNIXPERT: Diagnosis of Printer Problems

Michael Lessel
Harold Boley

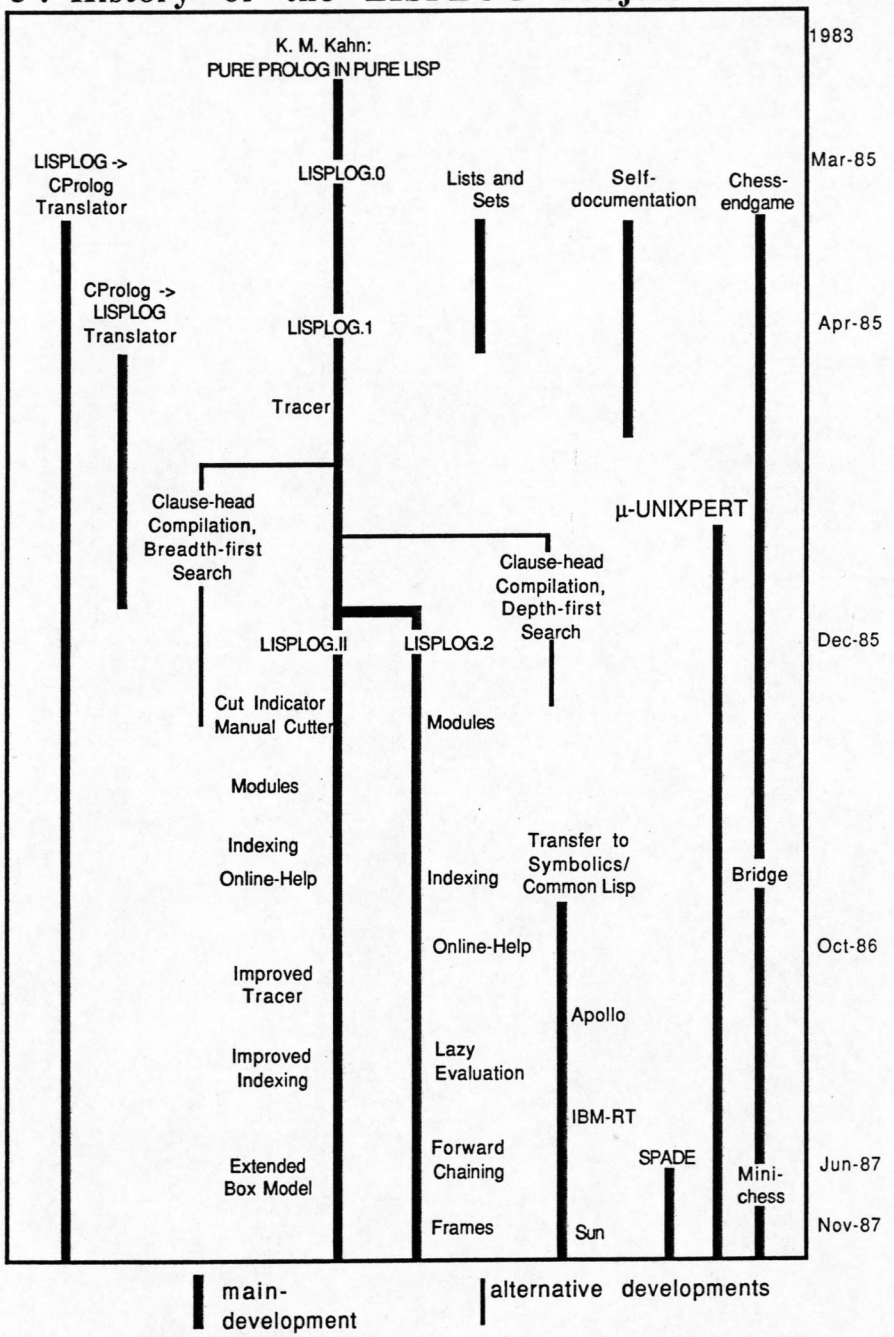
SEKI REPORT SR-87-09

August 1987

ABSTRACT

The μ-UNIXPERT systems perform knowledge-based diagnosis in the domain of printers as part of the computer periphery. They are implemented in LISPLOG, a functional/logical language that provided the required programming spectrum ranging from operating-system calls to the explanation component.

5. History of the LISPLOG Project



6. German-English glossary

Abflachen	flattening
Ableitung	derivation
Ausdruck	expression
Berechnung	computation
Beweis	proof
Durchgriff	access
Folgerung	inference
Funktionsaufruf	function application
Hand-Schneider	manual cutter
Loesung	solution
Muster	pattern
Regel	rule
Schachtelung	nesting
Schnitt-Anzeiger	cut indicator
Uebersetzung	translation
Umgebung	environment
Umwandlung	transformation
Vereinheitlichung	unification
Werkzeug	tool
Wert	value
Ziel	goal

7. The Syntax of LISPLOG

LISPLOG-Program	::=	{ *clause* | *function* }*
clause	::=	(*conclusion* { *premise* }*)
conclusion	::=	*cut-conclusion* | *simple-conclusion*
premise	::=	*predicate-call* | *function-call* | *primitive-call*
predicate-call	::=	(*predicate* { *argument* }* [. *variable*])
primitive-call	::=	(**is** *argument argument*) | (**not** *premise*)
predicate	::=	*predicate-constant* | *predicate-variable*
predicate-constant	::=	*LISP-atom*
predicate-variable	::=	*simple-variable* (must be bound when called)
cut-conclusion	::=	**!** *simple-conclusion*
simple-conclusion	::=	(*predicate-constant* { *argument* }* [. *variable*])
argument	::=	*variable* | *LISP-expression*
variable	::=	*anonymous-variable* | *simple-variable*
simple-variable	::=	**_** *LISP-atom*
anonymous-variable	::=	**ID**
LISP-expression	::=	*LISP-atom* | *list*
list	::=	({ *LISP-expression* }* [. *LISP-expression*])

LISP-atom may not contain **!**, **_** , or **?**

8. A bibliography on functional/logical integration

[Abramson 1984] H. Abramson: A Prological Definition of HASL: A Purely Functional Language with unification-based conditional Binding Expressions. New Generation Computing, Vol. 2, No. 1, pp. 3-35, 1984. Also in: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Allen et al. 1983] J. F. Allen, M. Giuliano, A. M. Frisch: The HORNE Reasoning System. University of Rochester, Computer Science Department, Rochester, NY 14627, TR 126, Dec. 1983

[Bailey 1985] D. Bailey: The University of Salford Lisp/Prolog System. Software - Practice and Experience, 15(6), Juni 1985, pp. 595-609

[Barbuti, Bellia, Levi 1986] R. Barbuti, M. Bellia, G. Levi: LEAF: A Language which integrates Logic, Equations and Functions. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Bellia, Levi 1986] M. Bellia, G. Levi: The Relation between Logic and Functional Languages: A Survey. Journal of Logic Programming, Vol. 3, No. 3, pp. 217-236, 1986.

[Bentley 1975] J. L. Bentley: Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9), Sep. 1975, pp. 509-517

[Bernardi 1986] A. Bernardi: Ein Indexierungskonzept fuer LISPLOG-Datenbasen. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP 86-10, Dezember 1986

[Bernardi 1987] A. Bernardi: LISPLOG: LISP/PROLOG-Vereinheitlichung in COMMON LISP auf IBM 6150. Vorfuehrung 63. IBM Hochschulkongress '87, Berlin, Juli 1987.

[Bernardi et al. 1987] A. Bernardi, M. Dahmen, M. Meyer: LISPLOG Benutzerhandbuch. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP 87-01, Januar 1987

[Bernardi & Kammermeier 1987] A. Bernardi, F. Kammermeier: LISPLOG.2: Implementation einer LISP/PROLOG-Vereinheitlichung. Vorstellung des erweiterten Systems in COMMON LISP. 4. Workshop Alternative Konzepte fuer Sprachen und Rechner, Bad Honnef, Maerz/April 1987. Universitaet Muenster, Angewandte Mathematik und Informatik, 2/87-1.

[Bobrow 1984] D. G. Bobrow: If PROLOG is the answer, what is the question? Proc. International Conference on Fifth Generation Computer Systems 1984, ICOT 1984, pp. 138-145

[Boley 1982/83] H. Boley: Artificial Intelligence Languages and Machines. Universitaet Hamburg, FB Informatik, IFI-HH-8-94/82, Dec. 1982. Also in: Technology and Science of Informatics, 2(3), Mai-Juni 1983, pp. 137-158

[Boley 1983] H. Boley: FIT - PROLOG: A Functional/Relational Language Comparison. Universitaet Kaiserslautern, FB Informatik, interner Bericht 95/83, MEMO SEKI-83-14, Dec. 1983

[Boley 1984] H. Boley: LISP - eine funktionale Einfuehrung. Universitaet Kaiserslautern, FB Informatik, MEMO SEKI-84-05, September 1984

[Boley 1986a] H. Boley: RELFUN: A Relational/Functional Integration with Valued Clauses. Universitaet Kaiserslautern, FB Informatik, SEKI Report SR-86-04, May 1986. Also in: SIGPLAN Notices 21(12), Dec. 1986, pp. 87-98

[Boley 1986b] H. Boley (Ed.): A Bird's-Eye View of LISP-LOG: The LISP/PROLOG Integration with Initial-Cut Tools. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-08, Dec. 1986

[Boley 1987a] H. Boley: LISP/PROLOG-Kombinatorik. mod-ki message 0043, June 1987.

[Boley 1987b] H. Boley: Fone and Fall: Forward-with-Backward Chaining in LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-03, June 1987

[Boley 1987c] H. Boley: Goal: Backward-with-Forward Chaining in LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-04, June 1987

[Boley 1987d] H. Boley: FIT: Declarative programming as transformer and adapter fitting. Universitaet Hamburg, FB Informatik, August 1987

[Boley 1987e] H. Boley: Frame and Heir: Clausal Frames and Multiple Inheritance in LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-09, November 1987

[Boley & Kammermeier et al. 1985] H. Boley, F. Kammermeier u. die LISPLOG-Gruppe: LISPLOG: Momentaufnahmen einer LISP/PROLOG-Vereinheitlichung. Universitaet Kaiserslautern, FB Informatik, MEMO SEKI-85-03, August 1985. Short version in: B. Nebel (Ed.): Papiere zum Workshop Logisches Programmieren und Lisp. TU Berlin, FB Informatik, KIT-REPORT 31, Dec. 1985, pp. 36-53

[Boley & Meyer 1986] H. Boley, M. Meyer: Implementation einer LISP/PROLOG-Vereinheitlichung und ihrer Interaktionsumgebung. In: F. Simon (Ed.): Implementierung von funktionalen und logischen Programmiersprachen, Bericht Nr. 8603, Institut fuer Informatik und Praktische Mathematik, Christian-Albrechts-Universitaet Kiel, Mai 1986.

[Boyer & Moore 1972] R. S. Boyer, J. S. Moore: The Sharing of Structure in Theorem Proving Programs. Machine Intelligence 7, Edinburgh 1972, pp. 101-116

[Bourgault et al. 1985] S. Bourgault, M. Dincbas, J. P. Le Pape: THE LISLOG SYSTEM. Centre National d'Etudes des Telecommunications, Note technique NT/LAA/SLC/186, Mai 1985

[Bruynooghe 1982] M. Bruynooghe: The Memory Management of PROLOG Implementations. In: K. Clark, S. A. Taernlund (Eds.): Logic programming. Academic Press, London, 1982, pp. 83-98

[Campbell & Hardy 1984] J. A. Campbell, S. Hardy: Should PROLOG be List or Record oriented. In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984

[Carlsson 1981] M. Carlsson: (Re)implementing PROLOG in LISP or YAQ - Yet Another QLOG. UPMAIL Technical Report 5, Uppsala University, Oktober 1981

[Carlsson 1985] M. Carlsson: A Microcoded Unifier for Lisp Machine Prolog. 1985 Symposium on Logic Programming, Juli 1984, Boston, Massachusetts, IEEE Computer Society Press, pp. 162-171

[Charniak & McDermott 1985] E. Charniak, D. McDermott: Introduction to Artificial Intelligence. Addison Wesley, 1985.

[Chester 1979] D. L. Chester: Using HCPRVR. Internal Report, Department of Computer Science, University of Texas at Austin, August 1979

[Chester 1980] D. Chester: HCPRVR: An Interpreter for Logic Programs. 1st NCAI-80, Stanford University, August 1980

[Clocksin & Mellish 1981/84] W. Clocksin & C. Mellish: Programming in PROLOG. Springer Verlag, Berlin Heidelberg New York, 1981. Second Edition 1984

[Cohen 1986] S. Cohen: The APPLOG Language. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Cohen & Feigenbaum 1982] P. R. Cohen, E. A. Feigenbaum: The Handbook of Artificial Intelligence, Volume III. Pitman, 1982

[Combuechen 1985] M. Combuechen: Symbolics to announce PROLOG Support for the 3600 Family of LISP Machines. SYMBO 10/84 E, European Symbolics Users Newsletter, 1(4), Januar 1985

[Dahmen 1985] M. Dahmen: Ein Translator von CPROLOG nach LISPLOG. In: [Dahmen, Herr, Hinkelmann, Morgenstern 1985]

[Dahmen 1986a] M. Dahmen: Iterativer LISPLOG Interpreter Implementierung, Dokumentation und Evaluation. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-03, Juni 1986

[Dahmen 1986b] M. Dahmen: LAZY-LISPLOG. Unveroeffentlichtes Manuskript, Universitaet Kaiserslautern, FB Informatik, 1986

[Dahmen, Herr, Hinkelmann, Morgenstern 1985] M. Dahmen, J. Herr, K. Hinkelmann, H. Morgenstern: LISPLOG: Beitraege zur LISP/PROLOG-Vereinheitlichung. Universitaet Kaiserslautern, FB Informatik, MEMO SEKI-85-10, November 1985

[Darlington, Field, Pull 1986] J. Darlington, A. J. Field, H. Pull: The Unifcation of Functional and Logic Languages. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Foderaro et al. 1983] J. K. Foderaro, K. L. Sklower, K. Laver: The FRANZ LISP Manual. University of California, Juni 1983

[Fogelholm 1984] R. Fogelholm: Exeter Prolog - some Thoughts on Prolog Design by a LISP User. In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984

[Fribourg 1984] L. Fribourg: Oriented Equational Clauses as a Programming Language. J. Logic Programming, 1984:2, pp. 165-177

[Frisch et al. 1983] A. M. Frisch, J. F. Allen, M. Giuliano: An Overview of the HORNE Logic Programming System. SIGART Newsletter, No. 84, April 1983, pp. 27-19

[Ghallab 1981] M. Ghallab: Decision Trees for Optimizing Pattern-Matching Algorithms in Production Systems. Proc. 7th IJCAI-81, Vancouver, Aug. 1981, pp. 310-312

[Goguen & Meseguer 1984] J. Goguen, J. Meseguer: Equal-

ity, Types, Modules, and Generics for Logic Programming. Stanford University, Center for the Study of Language and Information, Report No. CSLI-84-5, March 1984. Also in: J. Logic Programming, 1984:2, pp. 179-210 Also in: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Gray 1984] P.M.D. Gray: Logic, Algebra and Databases. Ellis Horwood Ltd., 1984

[Greussay 1983] P. Greussay: LOVLISP: Une extension de VLISP vers PROLOG. Documentation en ligne. Universite Paris-8 & L.I.T.P., August 1983. Also in: M. Dincbas (Ed.): Programmation en Logique. Actes du Seminaire 1983, Perros-Guirec, Maerz 1983

[Gross 1985] E. Gross: BABYLON-Prolog. In: B. Nebel (Ed.): Papiere zum Workshop Logisches Programmieren und Lisp. TU Berlin, FB Informatik, KIT-REPORT 31, Dec. 1985, pp. 30-35

[Hansson, Haridi, Taernlund 1982] A. Hansson, S. Haridi, S. A. Taernlund: Properties of a Logic Programming Language. In: K. Clark, S. A. Taernlund (Eds.): Logic Programming. Academic Press, London, 1982

[Herr 1985] J. Herr: Breitensuche und Klauselcompilation fuer LISPLOG. In: [Dahmen, Herr, Hinkelmann, Morgenstern 1985]

[Herr 1986] J. Herr: Ansatz fuer einen LISPLOG Compiler mit LISP als Zielsprache. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-06, Dezember 1986

[Herr 1988] J. Herr: SPADE: A Program Analysing the Stability of Finite Difference Methods for the Numerical Solution of Linear Partial Differential Equations With Constant Coefficients. Universitaet Kaiserslautern, Fachbereich Informatik, Diplomarbeit, erscheint

[Hinkelmann 1986] K. Hinkelmann: Uebersetzung von LISPLOG-Programmen nach CPROLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-05, September 1986. Erweiterte Fassung April 1987

[Hinkelmann 1988] K. Hinkelmann: SASLOG: Eine funktional-logische Sprachintegration mit Lazy Evaluation und semantischer Unifikation. Universitaet Kaiserslautern, FB Informatik, Diplomarbeit, erscheint

[Hinkelmann & Morgenstern 1985] K. Hinkelmann, H. Morgenstern: Ein Verfahren zur Transformation von LISP-Funktionen in PROLOG-Relationen. In: [Dahmen, Herr, Hinkelmann, Morgenstern 1985]

[Hinkelmann, Noekel, Rehbold 1988] K. Hinkelmann, K. Noekel, R. Rehbold: SASLOG: Lazy Evaluation Meets Backtracking. Universitaet Kaiserslautern, FB Informatik, erscheint

[Hoelldobler 1984] S. Hoelldobler: Functional and Logic Programming. Universitaet der Bundeswehr, FB Informatik, Muenchen, Bericht Nr. 8408, Juni 1984

[Hoelldobler, Furbach, Laussermair 1985] S. Hoelldobler, U. Furbach, T. Laussermair: Extended Unification and its Implementation. Universitaet der Bundeswehr, FB Informatik, Muenchen, Bericht Nr. 8504, Juni 1985

[JIPDEC 1980] Interim report on study and research on fifth-generation computers (outline). Japan Information Processing Development Center, 1980

[Kahn 1981] K. M. Kahn: UNIFORM - A Language based upon

Unification which unifies (much of) LISP, PROLOG, and ACT 1. Proc. 7th IJCAI-81, Vancouver, Aug. 1981, pp. 933-939

[Kahn 1982] K. M. Kahn: Unique Features of Lisp Machine Prolog. In: Procedures of the Workshop on PROLOG Programming Environments, Linkoeping, March 1982

[Kahn 1983] K. M. Kahn: Unique Features of Lisp Machine Prolog. UPMAIL Technical Report No. 15, Uppsala University, 14. Februar 1983

[Kahn 1983/84] K. M. Kahn: Pure PROLOG in Pure LISP. Logic Programming Newsletter 5, Winter 83/84, pp.3-4

[Kahn 1986] K. M. Kahn: UNIFORM - A Language based upon Unification which unifies (much of) LISP, PROLOG and ACT 1. (Revised version) In: Doug DeGroot & Gary Lindstrom (Eds.): Logic Programming - Functions, Relations and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986, pp. 411-438

[Kahn & Carlsson 1983] K. M. Kahn, M. Carlsson: LM-Prolog User Manual. UPMAIL, Department of Computing Science, Uppsala University, October 1983

[Kahn & Carlsson 1984] K. M. Kahn, M. Carlsson: How to implement Prolog on a LISP Machine. In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984

[Kammermeier 1984] F. Kammermeier: Franz-Lisp Mini Handbuch. Universitaet Kaiserslautern, FB Informatik, Juni 1984

[Kammermeier 1985] F. Kammermeier: Dokumentation der Prolog-Implementation in Franz-Lisp. Universitaet Kaiserslautern, FB Informatik, April 1985

[Kammermeier 1986] F. Kammermeier: LISPLOG im Kontext anderer LISP/PROLOG-Vereinheitlichungen. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-09, Dezember 1986

[Kluzniak & Szpakowicz 1985] F.Kluzniak, S. Szpakowicz, J. S. Bien (Contr.): Prolog for Programmers. Academic Press, London, 1985

[Knoepfler & Hotop 1985] S. Knoepfler, S. Hotop: ConProlog - Eine Prolog-Implementation in Interlisp-D. In: B. Nebel (Ed.): Papiere zum Workshop Logisches Programmieren und Lisp. TU Berlin, FB Informatik, KIT-REPORT 31, Dec. 1985, pp. 54-59

[Komorowski 1982] H. Komorowski: QLOG - The Programming Environment for PROLOG in LISP. In: K. Clark, S. A. Taernlund (Eds.): Logic programming. Academic Press, London, 1982, pp. 315-322

[Kornfeld 1983] W. Kornfeld: Equality for Prolog. Proc. 8th IJCAI-83, Karlsruhe, Aug. 1983, pp. 514-519

[Kowalski 1983] R. Kowalski: Logic Programming. In: Proc. IFIP, pp. 133-145, Amsterdam, North-Holland

[Kowalski 1985] R. Kowalski: The Relation between Logic Programming and Logic Specification. In: C.A.R. Hoare, J.L. Shepherdson: Mathematical Logic and Programming Languages, Prentice/Hall International, 1985

[Lee 1986] N. S. Lee: Programming with P-Shell. IEEE Expert 1(2), Summer 1986

[Lessel 1986] M. Lessel: micro-UNIXPERT: Ein wissensbasiertes System zur Behandlung von Problemen bei UNIX-Druckauftraegen. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-04, Mai 1986

[Lessel & Boley 1987] M. Lessel, H. Boley: micro-UNIXPERT: Diagnosis of Printer Problems. Universitaet Kaiserslautern, FB Informatik, SEKI Report SR-87-09, August 1987

[Mayer & Richter 1987] O. Mayer, M. M. Richter: Funktional/logische Expertensystemtools. Allgemeine Angaben zum Teilprojekt FLEX. SFB 314, Universitaet Kaiserslautern, FB Informatik, Januar 1987

[McDermott 1975] D. McDermott: Very Large PLANNER-Type Data Bases. MIT, AI Memo 339, Sep. 1975

[McDermott 1980] D. McDermott: The PROLOG Phenomenon. SIGART Newsletter, No. 72, Juli 1980, pp. 16-20

[Mellish 1982] C. S. Mellish: An Alternative to Structure Sharing in the Implementation of a PROLOG Interpreter. In: K. Clark, S. A. Taernlund (Eds.): Logic programming. Academic Press, London, 1982, pp. 99-106

[Meyer 1987] M. Meyer: Entwurf und Implementierung einer Interaktionsumgebung fuer LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-87-02, Januar 1987

[Moffat & Gray 1986] D. S. Moffat, P. M. D. Gray: Interfacing Prolog to a persistent Data Store. in: E. Shapiro (Ed): Third International Conference on Logic Programming. London, July 1986. LCMS 225, Springer-Verlag, 1986, pp. 577-584.

[Nilsson 1984] M. Nilsson: The World's Shortest Prolog Interpreter? In: J. A. Campbell (Ed.): Implementations of PROLOG. Ellis Horwood Ltd., 1984

[Noekel 1985] K. Noekel: SASL: Implementierung eines Reduktionsalgorithmus und Steuerung und Organisation eines Beweissystems fuer eine Logik. Diplomarbeit, RWTH Aachen, Dezember 1985

[Noekel & Rehbold 1986] K. Noekel, R. Rehbold: SASL: Implementierung einer rein funktionalen Sprache mit Lazy Evaluation. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-07, November 1986

[Okuno et al. 1984] H. G. Okuno, I. Takeuchi, N. Osato, Y. Hibino, K. Watanabe: TAO: A Fast Interpreter-Centered System on Lisp Machine ELIS. Conference Record of the 1984 ACM Symposium on LISP and Functional Programming, Austin, Texas, August 1984, pp. 140-149

[Pereira ohne Datum] F. Pereira (Ed.): CProlog User's Manual. University of Edinburgh, Dept. of Architecture

[Ramamohanaro & Shepherd 1986] K. Ramamohanaro, J. Shepherd: A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases. in: E. Shapiro (Ed): Third International Conference on Logic Programming. London, July 1986. LCMS 225, Springer-Verlag, 1986, pp. 569-576.

[Read & Dyer 1985] W. Read, M. G. Dyer: TLOG - Yet another Logic System in Lisp? Tech. Rep. UCLA-AI-85-1, 1985

[Reddy 1986] U. S. Reddy: On the Relationship between Logic and Functional Languages. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[Rehbold 1985] R. Rehbold: SASL: Implementierung eines Abstraktionsalgorithmus und Beweisalgorithmen fuer eine Logik. Diplomarbeit, RWTH Aachen, Dezember 1985

[Robinson 1986] J. Robinson: A Prolog Processor Based on a Pattern Matching Memory Device. in: E. Shapiro (Ed): Third International Conference on Logic Programming. London, July 1986. LCMS 225, Springer-Verlag, 1986, pp. 172-178.

[Robinson & Sibert 1981] J. Robinson, E. Sibert: The LOGLISP User's Manual. School of Computer and Information Science, Syracuse University, December 1981

[Robinson & Sibert 1982a] J. Robinson, E. Sibert: LOGLISP: Motivation, Design and Implementation. In: K. Clark, S. A. Taernlund (Eds.): Logic Programming. Academic Press, London, 1982, pp. 299-313

[Robinson & Sibert 1982b] J. Robinson, E. Sibert: LOGLISP: an alternative to PROLOG. Machine Intelligence 10, Chichester, 1982, pp. 399-419

[Rulifson et al. 1972] J. F. Rulifson, J. A. Derksen, R. J. Waldinger: QA4: A Procedural Calculus for Intuitive Reasoning. Menlo Park: SRI Technical Note 73, 1972

[SALFORD 1984] The University of Salford LISP/PROLOG Reference Manual. University of Salford, Second Edition, March 1984

[Sansonnet 1986] J. P. Sansonnet: The Machine for Artificial Intelligence Applications: MAIA. Proc. 7th ECAI-86, Volume I, Brighton, Juli 1986

[Sato & Sakurai 1983] M. Sato, T. Sakurai: Qute: A Prolog/Lisp Type Language for Logic Programming. Proc. 8th IJCAI-83, Karlsruhe, Aug. 1983, pp. 507-513

[Sato & Sakurai 1984] M. Sato, T. Sakurai: QUTE: A Functional Language based on Unification. Proc. International Conference on Fifth Generation Computer Systems 1984, ICOT 1984, pp. 157-165

[Schrag 1984] Robert C. Schrag: Compilation and Environment Optimisations for LOGLISP. Rome Air Developement Center, In-House Report RADC-TM-84-14, July 1984

[Subrahmanyam 1984] P. A. Subrahmanyam, J.-H. You: Conceptual Basis and Evaluation Strategies for Integrating Functional and Logic Programming. 1984 International Symposium on Logic Programming, Februar 1984, Atlantic City, New Jersey, IEEE Computer Society Press, pp. 144-153

[Subrahmanyam, You 1986] P. A. Subrahmanyam, J.-H. You: FUNLOG: A Computational Model integrating Logic Programming and Functional Programming. In: D. DeGroot & G. Lindstrom (Eds.): Logic Programming - Functions, Relations, and Equations. Prentice-Hall, Englewood Cliffs, NJ, 1986

[SYMBOLICS 1985] User's Guide to Symbolics Prolog. Symbolics Inc., Cambridge, Massachusetts, June 1985

[Takeuchi et al. 1983] I. Takeuchi, H. Okuno, N. Ohsato: TAO - A harmonic mean of Lisp, Prolog and Smalltalk. SIGPLAN Notices, V18 #7, Juli 1983, pp. 65-74

[VanEmden 1980] M. VanEmden: McDermott on Prolog: A Rejoinder. SIGART Newsletter, No. 73, Oktober 1980, pp. 19-20

[Voda & Yu 1984] P. J. Voda, B. Yu: RF-Maple: A Logic Programming Language with Functions, Types, and Concurrency. Proc. International Conference on Fifth Generation Computer Systems 1984, ICOT 1984, pp. 341-347

[Wallace 1983] R. S. Wallace: An Easy Implementation of PiL. SIGART Newsletter, No. 85, July 1983, pp. 29-32