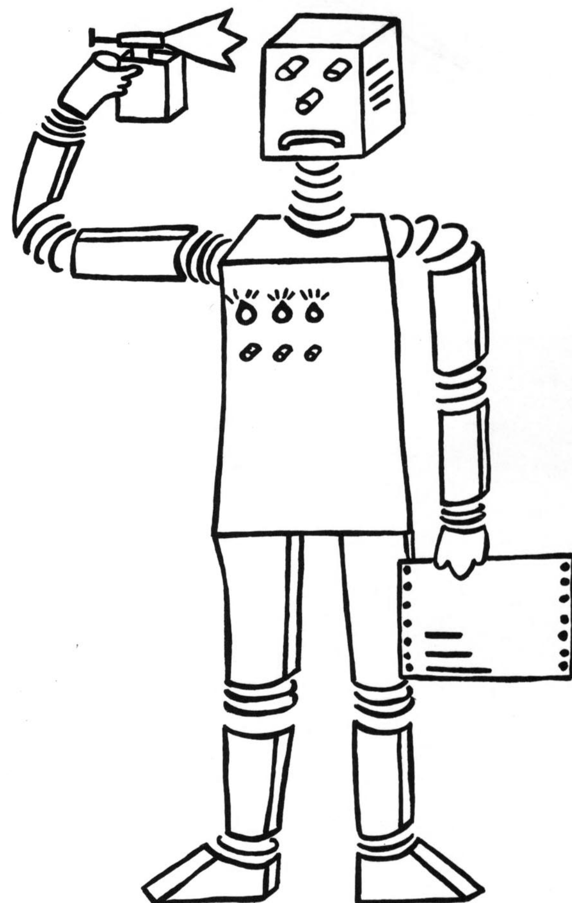
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

SEKI-Working Paper



A bird´s-eye view of LISPLOG:

The LISP/PROLOG integration with initial cut tools

Harold Boley (Editor)

November 1986 SWP-86-08

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 was supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 314, "Künstliche Intelligenz - Wissensbasierte Systeme"

General information about LISPLOG

Mail address:
LISPLOG-Projekt, Raum 14/403
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
Secretaries (a.m.)	:	-2802
(p.m.)	:	-2612

Current Members of the LISPLOG Project:

Coordination
Harold Boley

Performance
Iterative Interpreter
Michael Dahmen
Clause indexing
Ansgar Bernardi

Concepts
Modules
Michael Dahmen
Comparison with other Approaches
Franz Kammermeier

Interaction
Break/Trace/Cut Tools
Manfred Meyer

Application
micro-UNIXPERT
Michael Lessel

Translator
LISPLOG to CPROLOG
Knut Hinkelmann

Contents:

Chapter		page
1.	How to get the LISPLOG system and documentation	6
2.	LISPLOG: A LISP-based LISP/PROLOG integration	7
3.	A sample LISPLOG dialog	38
4.	Abstracts of the LISPLOG papers	49
5.	History of the LISPLOG project	61
6.	German-English glossary	62
7.	The syntax of LISPLOG	63
8.	A bibliography on LISP/PROLOG integration	64

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.

Copyright notice

This software and documentation is published for non-profit and research purposes only. We retain the exclusive right of its possible future commercial distribution.

Further non-profit distribution of the software is only permitted if (1) this notice is included and (2) the receiver´s address is sent to us in parallel. Please inform us about any modifications, improvements, and extensions of the software product.
This copyright note may not be changed.

The authors give no warranty of any kind for this product.
They would welcome suggestions for further debugging and improving the program, but cannot accept any obligation to follow such suggestions.

Copyright (c) 1986

™ UNIX is a registered trade-mark of Bell Laboratories

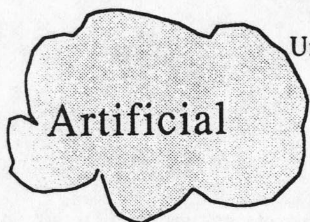
2. LISPLOG: A LISP-based LISP/PROLOG integration

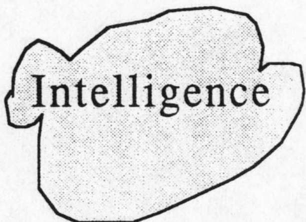
This chapter is a collection of overhead transparencies, originally prepared for an oral presentation of the LISPLOG project, on 6 November 1986.

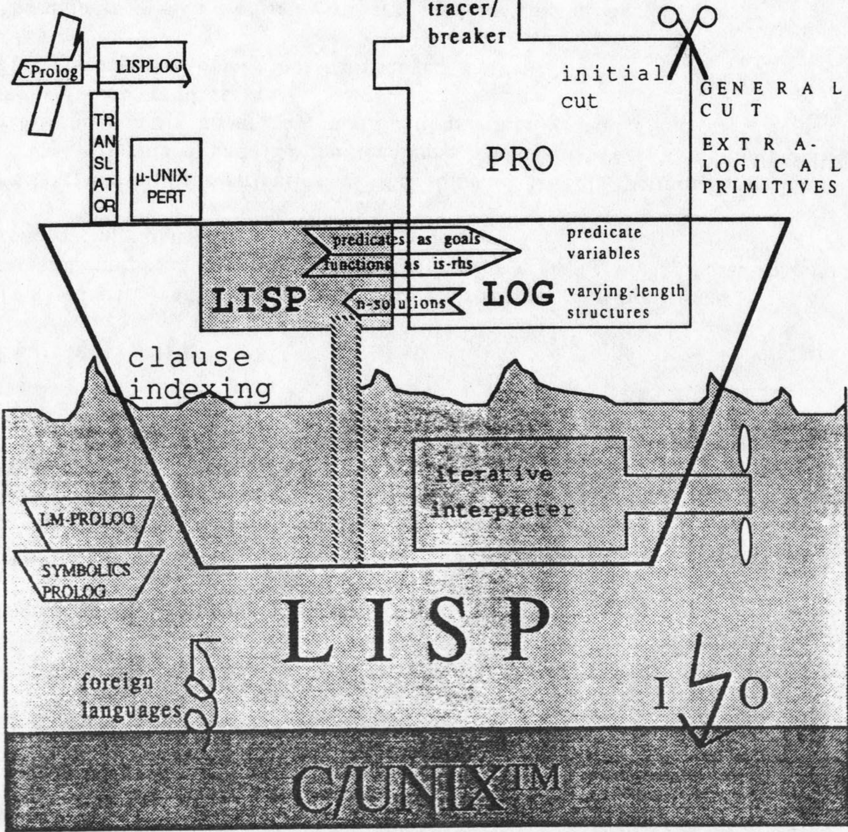
LISPLOG

A LISP-based LISP/PROLOG Integration

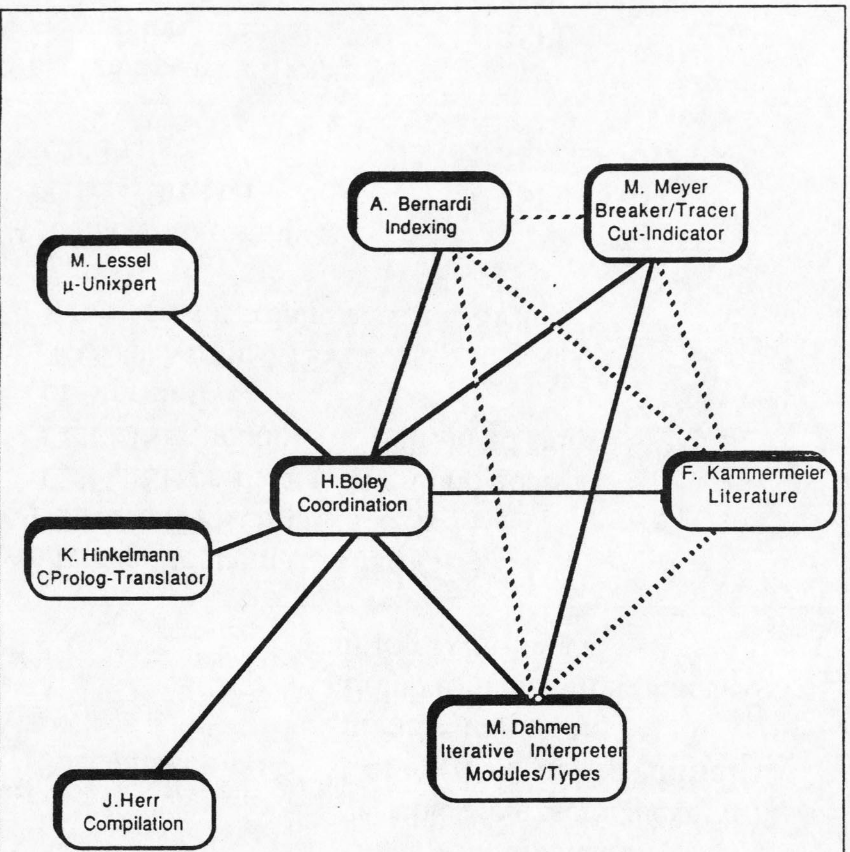
Harold Boley et al.
SFB 314
FB Informatik
Uni Kaiserslautern







The LISPLOG Team



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)

Finally improve efficiency :

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

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)
Types	(Mycroft/O´Keefe-style type inference)

- **Application** case studies :

μ-UNIXPERT	(diagnose UNIX printing problems)
μ-XPS I/II	(... *planning stage* ...)

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 Meyer

Programming Environment consists of:
- tracer (incl. cut-indicator, backtracing)
- break-package (debugger)
- stepper
- manual cutter

properties:
- optional
- automatically loaded when needed
- modularity (tracer/breaker)
- universality (same code for LISPLOG.2 and LISPLOG.II)
- integrated
- inefficient

How to trace a LISPLOG program ?

(1) use LISP Trace to watch the execution of the LISPLOG program:

advantage:	good for tracing functional part of the LISPLOG program
disadvantage:	useless for tracing execution of LISPLOG predicates

(2) use box model (by L.Byrd) instead:

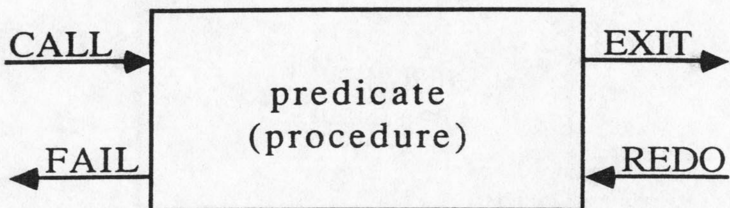
advantage:	good model for describing the operational semantics of PROLOG
disadvantage:	no model for the functional part of the LISPLOG program

(3) use an extended box model:

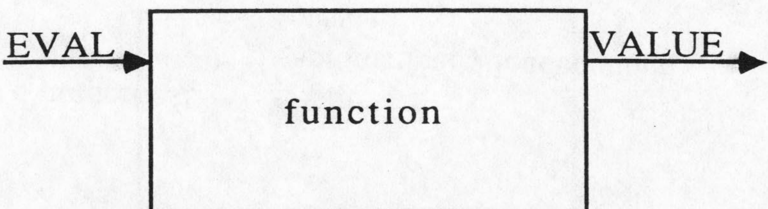
advantage:	integrated model for describing both: the functional and relational part of a LISPLOG program

The extended Box Model for LISPLOG

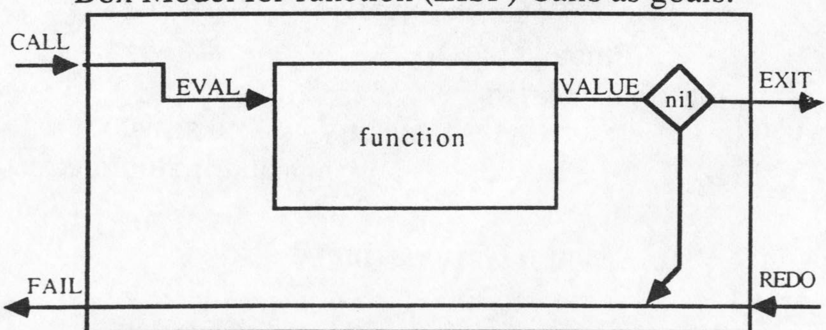
Box Model for predicates:



Box Model for functions:



Box Model for function (LISP) Calls as goals:



Trace Informations:

control-flow through the box-ports

results of unification:
(1) success: Δ unifikator = $\theta_{new} - \theta_{old}$
(2) failure: "mismatch"

backtracking:
"backtracked" variable-bindings
= last Δ unificator

module-switches:
see LISPLOG-Module-System [Dahmen 86]

initial-cut-indicator:
information about number of clauses cut

All trace informations are stored in the backtrace stack

⇓

full backtrace-possibility by paging through the backtrace stack

The LISPLOG Break-Package:

activated by breakpoints at some box-ports
or
activated by user-interrupt via CTRL-C
or
activated via manual-cutter

Break-Commands:

all LISPLOG-Toplevel-Commands !

backtracing

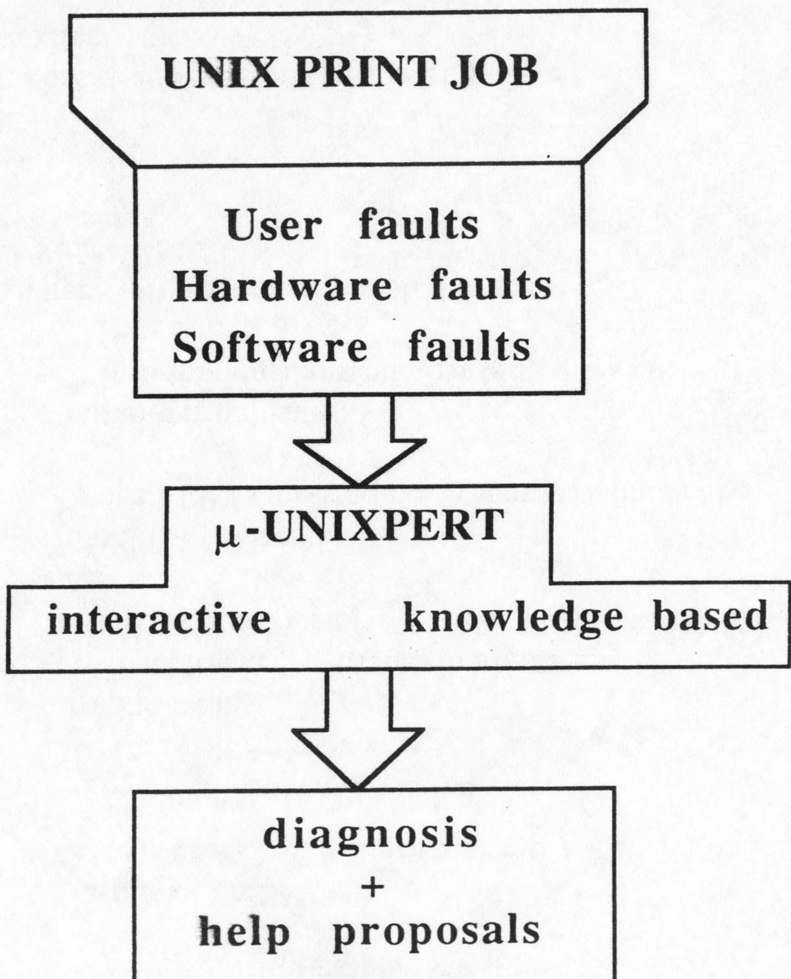
stepping

poking around in the actual LISPLOG-state (displaying goal, list-of-goals, actual path, etc.)

μ-UNIXPERT

(UNIX-PERiphery Tester)

Michael Lessel

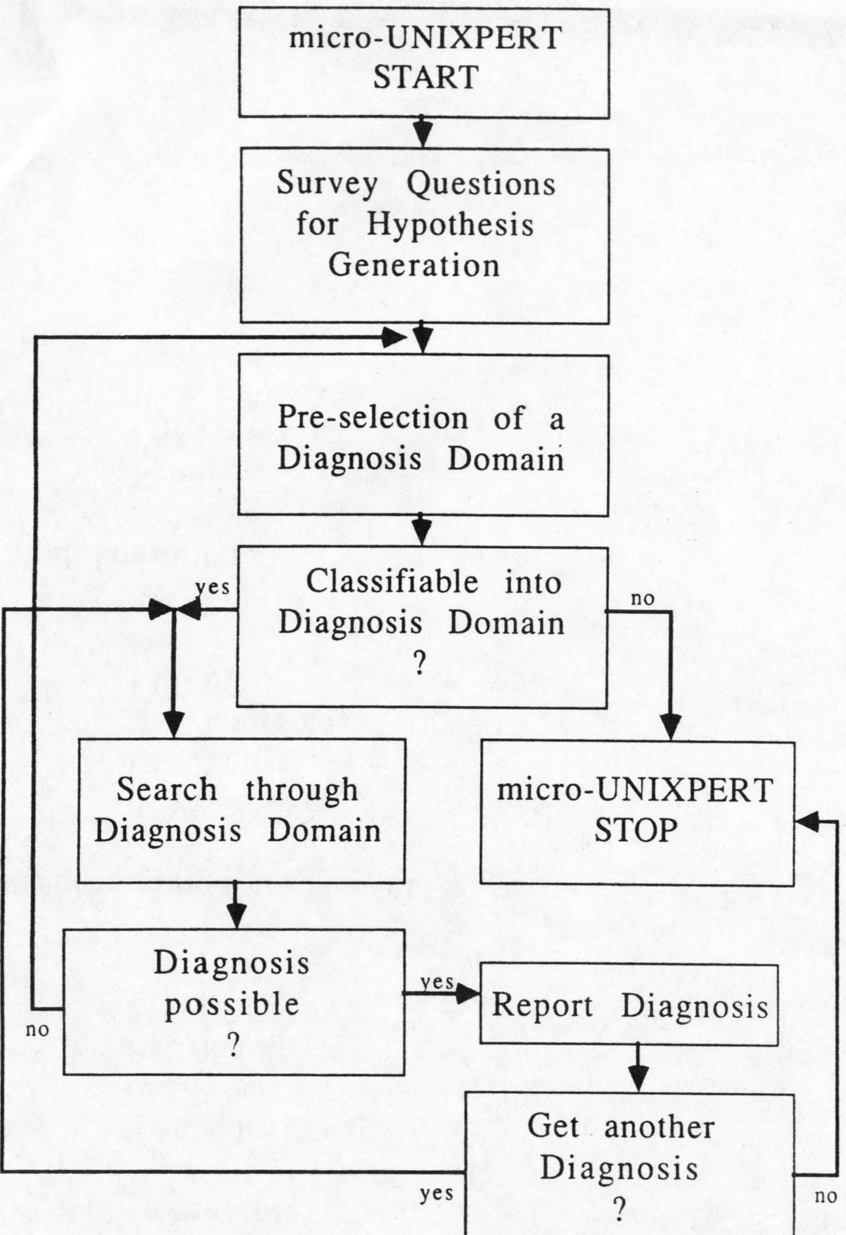


Why LISPLOG ?

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)
400 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

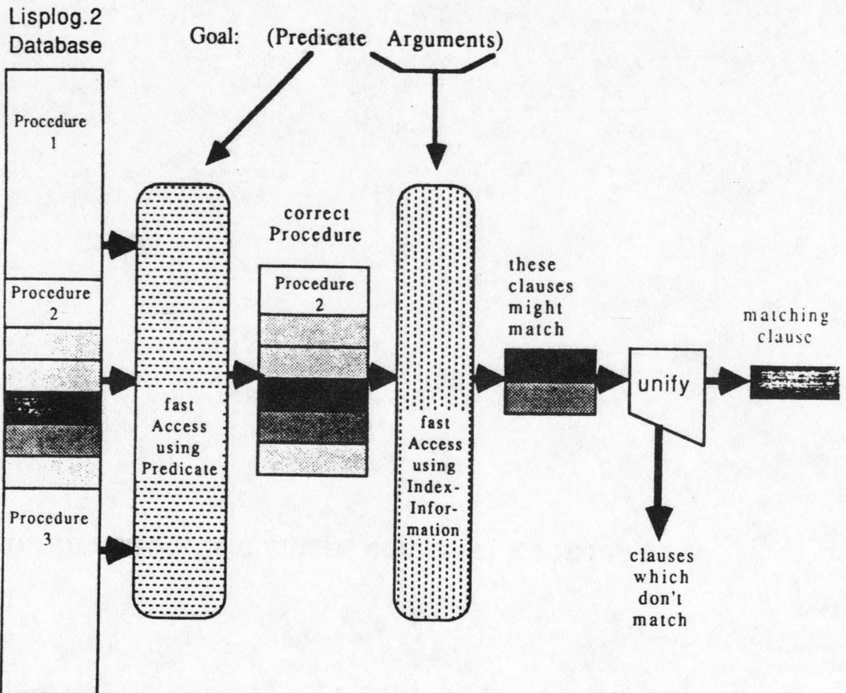
Indexing of Clauses

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 l)
          (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] ass-top (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] 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 the papers are in German, chapter 6 provides some help for translation.

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 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.

LISPLOG Benutzerhandbuch

Ansgar Bernardi, Michael Dahmen, Manfred Meyer

SEKI-Working-Paper 86-??

Dezember 1986

ABSTRACT

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

It gives a short description of the syntax of LISPLOG together with an explanation of all commands available on the toplevel 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.

LISPLOG

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".

ITERATIVER LISPLOG INTERPRETER

Implementierung, Dokumentation und Evaluation

Michael Dahmen

SEKI WORKING PAPER SWP-86-03

Juni 1986

ABSTRACT

This paper discusses the iterative interpreter LISPLOG.2, which was developed from the earlier recursive version LISPLOG.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.

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 relationcalls 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.

ÜBERSETZUNG VON LISPLOG-PROGRAMMEN NACH CPROLOG

Knut Hinkelmann

SEKI WORKING PAPER SWP-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.

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.

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

Michael Lessel

SEKI WORKING PAPER SWP-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

Entwurf und Implementierung einer Interaktionsumgebung fuer LISPLOG

SEKI WORKING PAPER SWP-86-??

Manfred A. Meyer

November 1986

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.

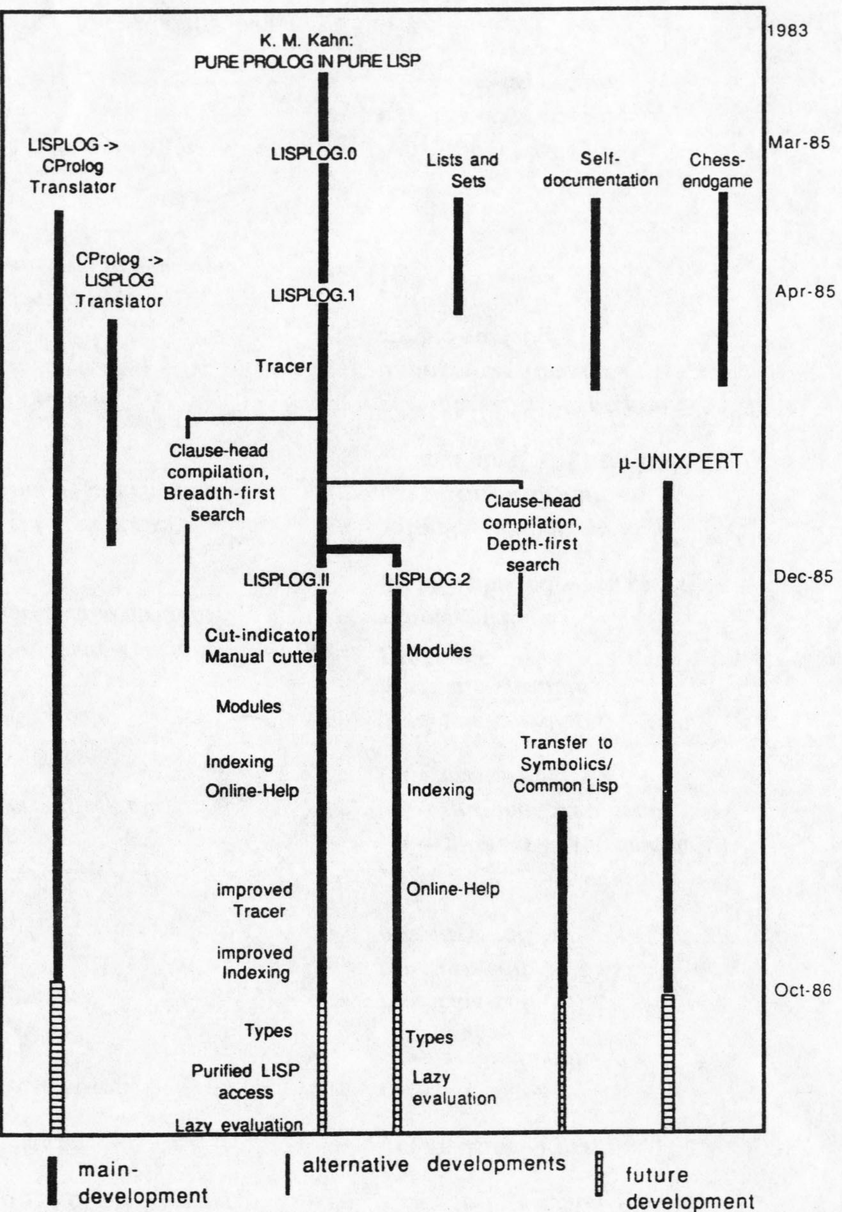
This programming environment consists of a box model tracer with backtrace possibility and a very comfortable break-package.

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.

5. History of the LISPLOG project



6. German-English glossary

Durchgriff	access
Folgerung	derivation
Regel	rule
Vereinheitlichung	unification
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 LISP/PROLOG integration

[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, S. 595-609

[Bernardi 1986] A. Bernardi: Ein Indexierungskonzept fuer LISPLOG-Datenbasen. Universitaet Kaiserslautern, FB Informantik, SEKI Working Paper SWP 86-??, November 1986

[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

[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 1986] H. Boley: RELFUN: A Relational/Functional Integration with Valued Clauses. Universitaet Kaiserslautern, FB Informatik, SEKI-REPORT SR-86-04, May 1986

[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.(Kurzfassung) In:F.Simon(Ed.):Implementierung von funktionalen und logischen Programmiersprachen, Bericht Nr.8603, Institut fuer Informatik und Praktische Mathematik, Christian-Albrechts-Univesitaet Kiel, Mai 1986

[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

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

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

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

[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

[Goguen & Meseguer 1984] J. Goguen, J. Meseguer: Equality, 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

[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

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

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

[Hinkelmann & Morgenstern 1985] K. Hinkelmann, H. Morgen-

stern: Ein Verfahren zur Transformation von LISP-Funktionen in PROLOG-Relationen. In: [Dahmen, Herr, Hinkelmann, Morgenstern 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, S. 933-939

[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

[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-??, November 1986

[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, S. 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

[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

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

[Meyer 1986] M. Meyer: Entwurf und Implementierung einer Interaktionsumgebung fuer LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper SWP-86-??, November 1986

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

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

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

[Robinson & Sibert 1982] 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

[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

[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

[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, S. 65-74

[VanEmden 1980] M. VanEmden: McDermott on Prolog: A Rejoinder. SIGART Newsletter, No. 73, Oktober 1980, S. 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