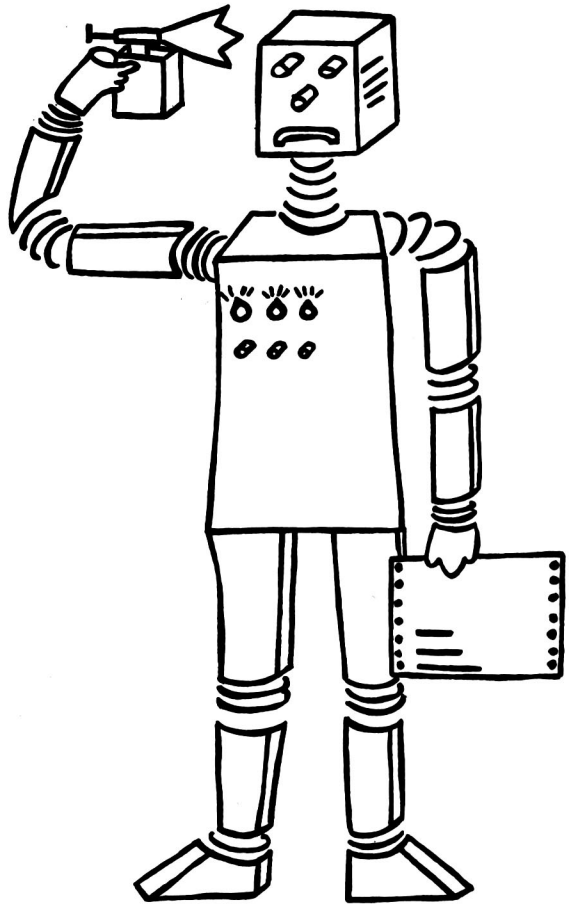
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

SEKI-Working Paper



Frame and Heir: Clausal Frames and Multiple Inheritance in LISPLOG

Harold Boley

SEKI Working Paper SWP-87-09

; **FRAME AND HEIR: CLAUSAL FRAMES AND MULTIPLE INHERITANCE IN LISPLOG**

; Harold Boley, FB Informatik, Univ. 675 Kaiserslautern, Box 3049, W. Germany

; SEKI Working Paper SWP-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.

; In two previous pages 'forward extensions' of LISPLOG's basic backward
; machinery were presented [Boley 1987b 1987c]. Here we complement these
; 'productions' by another knowledge-representation formalism, a version of
; 'frames'. Taken together, these LISPLOG extensions can be regarded as the
; kernel of a simple 'expert-system shell', LLshLL.0 (LispLog sheLL no. 0),
; based primarily on the PROLOG part of our functional/logical language.

; The 'frames extensions' considered here consist of a translator, frame,
; and an interpreter, heir:
; * frame generates a set of clauses from a frame description of an object,
; with attributes of slots becoming predicates (relations) of conclusions,
; and the object becoming copied into their first argument position;
; * heir proves an arbitrary goal by first trying it as an ordinary LISPLOG
; goal, and then trying to find an ako link from its first argument to any
; superobject and continuing recursively with a new goal that employs this
; superobject as a substitute for its first argument.
; The frame and heir parts can be used in isolation as well as together.

; Appendix III shows some facts and rules in the notation of LISPLOG's usual
; clausal style; appendix IV augments these by some frames in the style of
; the frame extension. Both can be consulted into the same database because
; the clauses asserted via frame calls merge naturally with clauses asserted
; directly. For example, the binary size relation created by the directly
; asserted fact (size canary small) in appendix III can be augmented by the
; frame-generated fact (size mouse small) in appendix IV. Thus, frame may be
; regarded as a kind of ass usable for groups of clauses involving the same
; object as their first argument. Once asserted, the querying of all kinds
; of clauses is done in the uniform notation of LISPLOG goals. For example,
; while it is unusual for frames to keep a slot's attribute and value fixed
; and asking for all objects characterized by it, after our translation a
; query such as (size _obj small) binds _obj to canary and then to mouse,
; without any difficulty. Let us remark here that our translator takes an
; 'object-centered' representation (frames) to produce a 'non-centered' one
; (clauses), from which LISPLOG.2 immediately produces a 'relation-centered'
; representation ('PROLOG procedures' as shown, e.g., by the listing command
; through the grouping of clauses around predicates); the inverse direction
; of translation (we might call it a 'frame decompiler') may also be useful,
; but would have to perform a partial mapping from the subset of clauses or
; relations producible by frame (e.g. excluding most kinds of rules).

; While slots of frames normally correspond to binary relations, we allow
; them to correspond to, and translate into, relations of arbitrary arities,
; exploiting LISPLOG's "."-notation for variable-length terms; for example,
; the goal (color bird . _which) succeeds with _which=(blue green yellow).
; [The animal taxonomy in appendices III and IV employs relations of arity
; greater than 2 for enumerating some default values of attributes like
; color, but we can also employ, say, a ternary sell relation in the more
; logical manner, as in the mini frame (frame mary (sell auto john)).]
; In particular, slots consisting only of an attribute without any value
; translate to unary relations; for example, (frame cat ... (clever) ...)

```
; becomes (ass (clever cat)), and can be queried by (clever cat . _w) with
; _w=nil or by (clever cat). The well-known binary isa or ako relation of
; semantic networks and frames, which we kept here (see below), may really
; be an artifact of the lack of unary relations in these formalisms: One
; might want to inherit to cat the properties of clever beings as well as
; those reachable via ako (mammal and pet), or indeed generally replace
; (frame ... (ako superobject) ...) by (frame ... (superobject) ...) and
; use all unary slots, (superobject), for inheritance. In any case, the
; possibility of 'variable-length slots' in frames appears to be useful.
; Another generalization that our frame concept inherits simply from its
; implementation in a logical programming language is the permission of
; non-ground slot values, in particular variables. In the transformation
; of an obj frame (frame obj ... (attr val1 val2 ... valN) ...) into a
; sequence of obj clauses, ... (ass (attr obj val1 val2 ... valN)) ...,
; each valI can either be ground or non-ground, whereas obj is normally
; ground (if a frame describes a fixed object) and attr must always be
; even a LISP atom (in LISPLOG.2 atoms are used for relation indexing).
; A simple example is the last slot of the mammal frame in appendix IV,
; which translates to the clause (ass (eat mammal _x-0)). [Of course, its
; interpretation, "mammals eat everything", would be overly general for
; a biologist, and it is even more clear that indiscriminate inheritance
; of such 'almost-universal' facts via ako links need not always lead to
; satisfactory results.]

; Our frames include a concept of procedural attachment more accurately
; called 'goal attachment': The dummy slot value "@" marks a 'procedure'
; (a list of goals) for computing the actual slot value, using "@" as a
; distinguished result variable. More precisely, a slot of the form
; (attr @ . . . (pred ... @ ...) . . .) can be logically interpreted as
; (attr (epsilon (v) . . . & (pred ... v ...) & . . .)), where a version
; of Hilbert's epsilon operator is used to denote one of the objects v
; for which the pred goal conjunction holds. For example, the last slot of
; the mouse frame in appendix IV (with two "@"-goals) can be interpreted as
; (flee (epsilon (v) (size v medium) & (ako v animal))). Our translator
; generates a LISPLOG rule for each such slot, in the example leading to
; (ass (flee mouse _v-10) (size _v-10 medium) (ako _v-10 animal)). [The
; variable _v becomes renamed to _v-10 through the dynamic ass execution
; on 'deduction level' 10.] Obviously, this attachment concept realizes
; only "if-needed procedures", which directly map into rules evaluated by
; the backward mechanism of LISPLOG. Our forward extensions cited above
; could be used for realizing "if-added procedures", even though these
; would also suggest the introduction of a class/instance distinction not
; utilized in the current frame concept.

; The ako ("a kind of") relation can be employed like any other relation
; (as in the second goal above), but it is distinguished by its special use
; inside the inheritance interpreter heir: If the goal (_attr _obj . _vals),
; given to heir, fails as an ordinary LISPLOG goal, heir enumerates the ako
; superconcepts, _sup, of its first argument, _obj, and then calls itself
; for (_attr _sup . _vals), in the PROLOG-usual depth-first fashion. For
; example, the heir-less goal (inhabit canary house) clearly fails since
; no inhabit relation can be deduced for canary in the ordinary LISPLOG
; manner; however, calling the heir interpreter with this same goal, i.e.
; (heir (inhabit canary house)), succeeds via (heir (inhabit pet house)),
; exploiting the 'sort information' (ako canary pet). The user has the
; responsibility for ordering multiple ako facts in an 'optimal' fashion
; (just as for ordering any other clauses). So in the previous example we
; had to pay for ordering the pet superconcept after the bird superconcept,
; because this caused the depth-first strategy to unnecessarily explore the
; bird hierarchy for non-existent inhabit information. Note that the ako
; links form 'subtype hierarchies', used by heir to replace objects by their
; superobjects within goals. These same hierarchies could also be used by
; many object-level rules (instead of one heir interpreter) that explicitly
; prove ako goals over logical variables, as exemplified in appendix III by
; the rule (ass (inhabit _x house) (ako _x human)) as opposed to the fact
; (ass (inhabit human house)); however, while this is sufficient for simple
```

```
; goals like (inhabit child house), for multi-level hierarchies it would
; require an explicit formulation of the transitivity of the ako relation,
; a potential source of non-termination if done naively.

; Perhaps the main advantage of inheritance systems like heir is the economy
; permitted by storing general information with the highest superconcept to
; which it applies and still allowing exceptional information to be stored
; for certain subconcepts. A well-known example is the slot (inhabit land)
; stored with the mammal frame and the slot (inhabit sea) stored for the
; whale frame in appendix IV: Requests like (heir (inhabit mouse _what))
; and (heir (inhabit cat _what)) bind _what to land via mammal inheritance,
; and (heir (inhabit whale _what)) binds _what to sea directly [as usual,
; the whale request will also mammal-inherit _what=land if pressed for a
; 'second answer' via LISPLOG's more command].

; Of course, ako links need not stem from frames but can be directly input
; as facts; however, the usual convention is to introduce ako links as the
; first slots of every frame.
; Another optional interaction between frame and heir is the use of heir
; calls in one or more goals of a slot's goal attachment. For example, our
; previous goal attachment could not be queried like the explicit mouse slot
; (flee cat) because (ako @ animal) will not succeed for cat; on the other
; hand, the cat slot (chase @ (heir (size @ small)) (heir (ako @ animal)))
; is queryable like (chase mouse) because (heir (size @ small)) succeeds
; (trivially) for mouse and (heir (ako @ animal)) succeeds (non-trivially)
; for the same object by going up one step to its mammal superobject.

; The reader may consult appendices I and II for complete details of the
; frame and heir implementations. It should be noted that the second clause
; of frame employs LISPLOG's is primitive to call FRANZ LISP's function
; subst to replace "@" by "_v" on all levels of a rule generated by goal
; attachment. The extra def macro definition of frame serves only to permit
; frame to operate also (as a LISP function call) during consult commands,
; i.e. to allow reading in frames from files. For instance, the reader is
; encouraged to consult this file into LISPLOG, perform a listing of the
; clauses read into the database, compare the frame-generated clauses with
; the frames in appendix IV, query them with examples as given in the body
; of this paper (perhaps tracing the computations using spy), and adding
; other frames interactively.


; References (orders: lisplog@uklirb.UUCP)

; [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


; Appendix I: The frame-to-clause translator

(ass (frame _obj))
(ass (frame _obj (_attr _mark . _goals) . _slots)
     (eq _mark @)
     (is _assert (subst _v @ (list (_attr _obj @) . _goals)))
     (ass . _assert)
     (frame _obj . _slots))
(ass (frame _obj (_attr . _vals) . _slots)
     (or (null _vals) (neq (car _vals) @))
     (ass (_attr _obj . _vals))
     (frame _obj . _slots))
(def frame (macro (frm) (list 'n-solutions (list 'quote frm) 1)))
```

```
; Appendix II: The ako-inheritance interpreter

(ass (heir _goal) _goal)
(ass (heir (_attr _obj . _vals))
     (ako _obj _sup)
     (heir (_attr _sup . _vals)))


; Appendix III: Some sample clauses

(ass (ako canary bird))
(ass (ako canary pet))
(ass (ako budgerigar bird))
(ass (ako child human))

(ass (size canary small))

(ass (sound canary warble) (happy canary))

(ass (can bird fly))

(ass (color canary yellow))
(ass (color bird blue green yellow))

(ass (inhabit pet house))
(ass (inhabit _x house) (ako _x human))


; Appendix IV: Some sample frames

(frame mouse
 (ako mammal)
 (size small)
 (color gray)
 (flee @ (size @ medium) (ako @ animal)))

(frame cat
 (ako mammal)
 (ako pet)
 (size medium)
 (sound meow spit)
 (clever)
 (color black brown gray white)
 (chase @ (heir (size @ small)) (heir (ako @ animal)))
 (flee dog))

(frame whale
 (ako mammal)
 (inhabit sea))

(frame mammal
 (ako animal)
 (inhabit land)
 (suckle young)
 (eat _x))
```