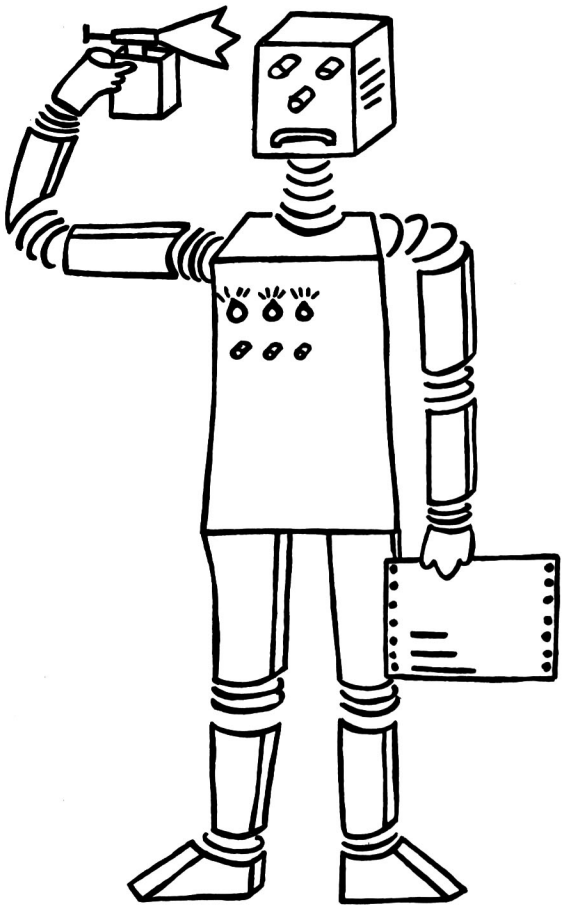
SEKI-Working Paper

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



Goal: Backward-with -Forward Chaining in LISPLOG

Harold Boley

SEKI WORKING PAPER SWP-87-04

```
; GOAL: BACKWARD-WITH-FORWARD CHAINING IN LISPLOG

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

; SEKI Working Paper SWP-87-04, June 1987


; Abstract: A tiny extension for perfoming forward chaining to prove goals
;           set up by LISPLOG's backward-chaining mechanism is introduced.


; This extension of LISPLOG realizes forward 'goal statements' derived
; from the fone (forward one) construct in [Boley 1987]: While fone pauses
; after each forward step to let the user decide (via the more command)
; whether enough assertions have been derived, goal calls an extra goal
; parameter for the same purpose. More precisely, goal proves a goal by
; first calling it as an ordinary LISPLOG goal (thus allowing arbitrary
; backward chaining), but on failure performs one step in the deduction
; cycle of a given forward production system, and then tries again.
; A production system is represented by LISPLOG rules with heads used
; for identifying the system and bodies whose conjunctions are divided
; in a (production-)premises part and a (production-)conclusion part.
; This representation is exactly the same as in [Boley 1987], which is
; similar to that in [Lee 1986]; but we describe only a special case:
; Here, we will use productions of the form (ass (s) p1 ... pN (nap c)),
; with pI as premises and c as conclusion; nap [read "not? assert! pp!"]
; asserts and pretty prints its argument iff it is not yet asserted nor
; provable. The sample systems a-e of [Boley 1987] can all be used via
; goal calls. For instance, system a below may be used by the goal call
; (goal (risky vinegar) (a)), which cannot prove (risky vinegar) in a
; purely backward manner, thus activates forward chaining by (a), until
; (risky vinegar) has become a (permanently available) fact. Of course,
; some backward steps using rules like (ass (avoid jane _x) (risky _x))
; may be required to access the results of forward steps activated by
; goals like (goal (avoid jane vinegar) (a)). Note that backward rules
; with goal premises like (ass (avoid john _x) (goal (risky _x) (a)))
; called by (avoid john vinegar) combine the chaining directions in a
; more efficient manner. Finally, the forward chaining activated by goal
; calls such as (goal (likes john _p) (b))) in the backward rule below
; may again employ some backward chaining for verifying a premise of a
; production such as (likes _x food) in system b, and so on:
(ass (warn john _p _o) (goal (risky _o) (a)) (goal (likes john _p) (b)))

; References (order [Boley 1987] and more LISPLOG papers: lisplog@uklirb.UUCP):
; [Boley 1986] H. Boley (Ed.): A Bird's-Eye View of LISPLOG: The LISP/PROLOG
; Integration with Initial-Cut Tools. Universitaet Kaiserslautern,
; FB Informatik, SEKI Working Paper SWP-86-08, Dec. 1986
; [Boley 1987] H. Boley: Fone and Fall: Forward-with-Backward Chaining in
; LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper
; SWP-87-03, June 1987
; [Lee 1986] N. S. Lee: Programming with P-Shell. IEEE Expert 1(2), Summer 1986

; The backward-with-forward implementation:
(ass (goal _go _sy) _go)                                    ; go all backward
(ass (goal _go _sy) (n-solutions _sy 1) (goal _go _sy))     ; sy step forward
(ass (nap _x) (not _x) (ass _x) (pp-external-form _x))      ; note 'dynamic ass'

; System a shows a depth-2 forward chaining acid->corrodent->risky:
(ass (a) (corrodent _x) (nap (risky _x)))          ; N=1
(ass (a) (acid _x) (nap (corrodent _x)))           ; N=1
(ass (a) (acid _x) (nap (piquant _x)))             ; N=1
(ass (acid vinegar))                               ; 'working memory' fact

; System b exemplifies a backward rule for verifying food liking:
(ass (b) (likes _x wine) (likes _x food) (nap (likes john _x)))    ; N=2
(ass (likes mary wine))                            ; 'working memory' fact 1
(ass (likes _y food) (corpulent _y))               ; 'working memory' rule
(ass (corpulent mary))                             ; 'working memory' fact 2
```

Goal: Backward-with -Forward Chaining in LISPLOG

Harold Boley

SEKI WORKING PAPER SWP-87-04

```
; GOAL: BACKWARD-WITH-FORWARD CHAINING IN LISPLOG

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

; SEKI Working Paper SWP-87-04, June 1987


; Abstract: A tiny extension for perfoming forward chaining to prove goals
;           set up by LISPLOG's backward-chaining mechanism is introduced.


; This extension of LISPLOG realizes forward 'goal statements' derived
; from the fone (forward one) construct in [Boley 1987]: While fone pauses
; after each forward step to let the user decide (via the more command)
; whether enough assertions have been derived, goal calls an extra goal
; parameter for the same purpose. More precisely, goal proves a goal by
; first calling it as an ordinary LISPLOG goal (thus allowing arbitrary
; backward chaining), but on failure performs one step in the deduction
; cycle of a given forward production system, and then tries again.
; A production system is represented by LISPLOG rules with heads used
; for identifying the system and bodies whose conjunctions are divided
; in a (production-)premises part and a (production-)conclusion part.
; This representation is exactly the same as in [Boley 1987], which is
; similar to that in [Lee 1986]; but we describe only a special case:
; Here, we will use productions of the form (ass (s) p1 ... pN (nap c)),
; with pI as premises and c as conclusion; nap [read "not? assert! pp!"]
; asserts and pretty prints its argument iff it is not yet asserted nor
; provable. The sample systems a-e of [Boley 1987] can all be used via
; goal calls. For instance, system a below may be used by the goal call
; (goal (risky vinegar) (a)), which cannot prove (risky vinegar) in a
; purely backward manner, thus activates forward chaining by (a), until
; (risky vinegar) has become a (permanently available) fact. Of course,
; some backward steps using rules like (ass (avoid jane _x) (risky _x))
; may be required to access the results of forward steps activated by
; goals like (goal (avoid jane vinegar) (a)). Note that backward rules
; with goal premises like (ass (avoid john _x) (goal (risky _x) (a)))
; called by (avoid john vinegar) combine the chaining directions in a
; more efficient manner. Finally, the forward chaining activated by goal
; calls such as (goal (likes john _p) (b))) in the backward rule below
; may again employ some backward chaining for verifying a premise of a
; production such as (likes _x food) in system b, and so on:
(ass (warn john _p _o) (goal (risky _o) (a)) (goal (likes john _p) (b)))

; References (order [Boley 1987] and more LISPLOG papers: lisplog@uklirb.UUCP):
; [Boley 1986] H. Boley (Ed.): A Bird's-Eye View of LISPLOG: The LISP/PROLOG
; Integration with Initial-Cut Tools. Universitaet Kaiserslautern,
; FB Informatik, SEKI Working Paper SWP-86-08, Dec. 1986
; [Boley 1987] H. Boley: Fone and Fall: Forward-with-Backward Chaining in
; LISPLOG. Universitaet Kaiserslautern, FB Informatik, SEKI Working Paper
; SWP-87-03, June 1987
; [Lee 1986] N. S. Lee: Programming with P-Shell. IEEE Expert 1(2), Summer 1986

; The backward-with-forward implementation:
(ass (goal _go _sy) _go)                                    ; go all backward
(ass (goal _go _sy) (n-solutions _sy 1) (goal _go _sy))     ; sy step forward
(ass (nap _x) (not _x) (ass _x) (pp-external-form _x))      ; note 'dynamic ass'

; System a shows a depth-2 forward chaining acid->corrodent->risky:
(ass (a) (corrodent _x) (nap (risky _x)))          ; N=1
(ass (a) (acid _x) (nap (corrodent _x)))           ; N=1
(ass (a) (acid _x) (nap (piquant _x)))             ; N=1
(ass (acid vinegar))                               ; 'working memory' fact

; System b exemplifies a backward rule for verifying food liking:
(ass (b) (likes _x wine) (likes _x food) (nap (likes john _x)))    ; N=2
(ass (likes mary wine))                            ; 'working memory' fact 1
(ass (likes _y food) (corpulent _y))               ; 'working memory' rule
(ass (corpulent mary))                             ; 'working memory' fact 2
```