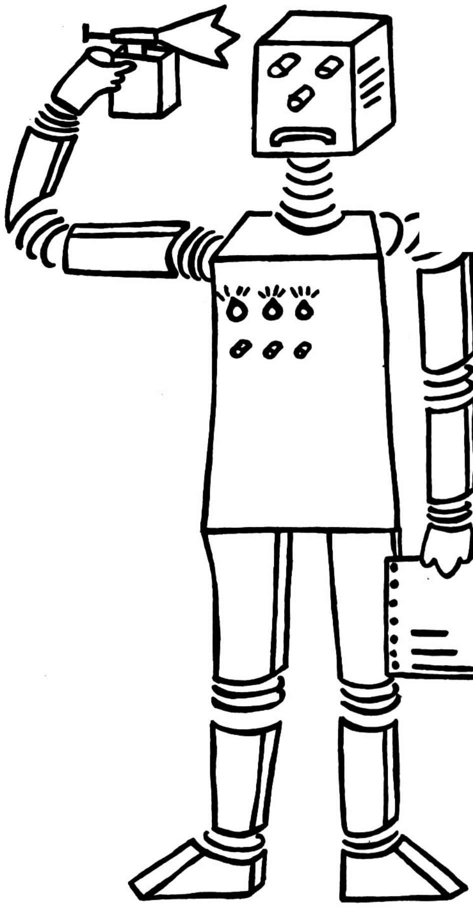
SEKI Working Paper

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



FONE AND FALL: Forward-with-Backward Chaining in LISPLOG

HAROLD BOLEY

SEKI WORKING PAPER SWP-87-03

```
; FONE AND FALL: FORWARD-WITH-BACKWARD CHAINING IN LISPLOG

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

; SEKI Working Paper SWP-87-03, June 1987


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


; This extension of LISPLOG realizes forward-computation 'productions'
; on the top-level, but permits backward-verification 'rules' (or, any
; LISPLOG programs) for proving the premises of productions.
; Productions come in groups related to the contexts in [Lee 1986].
; Such a production 'system' s is called by fone/fall (forward one/all)
; with (s ...) as argument; thus, (s) is usable as a degenerated pattern
; that constitutes the head of all LISPLOG rules representing the system
; (with larger patterns, production-system usage can be parameterized).
; The deduction cycle of fone calls is controlled by backtracking, i.e.
; it proceeds in a single-step fashion governed by LISPLOG's more command.
; (n-solutions ... 1) avoids final cuts for all productions of all systems.
; A production of system s is notated by (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. A sample system like a below may be used by typing (fone (a)),
; followed by more, ... or typing (fall (a)); however, (fall (d)) diverges.
; For system c a trace with the spy command can be instructive.

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

; The forward-with-backward implementation:
(ass (fone _sy) (n-solutions _sy 1) (forward one _sy))  ; one step at a time
(ass (fall _sy) (not (forward all _sy)))                ; all steps together
(ass (forward one _sy))                                  ;reflexive  and
(ass (forward _x _sy) (n-solutions _sy 1) (forward _x _sy)) ;transitive closure
(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

; System c uses a conclusion containing an anonymous [ID] variable:
(ass (c) (ok ich) (ok du) (nap (roger ID)))        ; N=2 ['alles Roger!']
(ass (c) (sonntagskind _x) (nap (ok _x)))          ; N=1
(ass (sonntagskind du))                            ; 'working memory' fact 1
(ass (ok ich))                                     ; 'working memory' fact 2

; System d demonstrates an infinite transitive closure enumerable by fone:
(ass (d) (natural _x) (nap (natural (succ _x))))  ; N=1 [recursive production]
(ass (natural 0))                                  ; 'working memory' fact

; System e employs predicate parameters filled via (e parent brother uncle):
(ass (e _p _q _r) (_p _x _y) (_q _y _z) (nap (_r _x _z)))     ; N=2
(ass (parent nina gina))                           ; 'working memory' fact 1
(ass (brother gina tino))                          ; 'working memory' fact 2
```

FONE AND FALL:
Forward-with-Backward Chaining in LISPLOG

HAROLD BOLEY

SEKI WORKING PAPER SWP-87-03

; FONE AND FALL: FORWARD-WITH-BACKWARD CHAINING IN LISPLOG

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

; SEKI Working Paper SWP-87-03, June 1987

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

```
; This extension of LISPLOG realizes forward-computation 'productions'
; on the top-level, but permits backward-verification 'rules' (or, any
; LISPLOG programs) for proving the premises of productions.
; Productions come in groups related to the contexts in [Lee 1986].
; Such a production 'system' s is called by fone/fall (forward one/all)
; with (s ...) as argument; thus, (s) is usable as a degenerated pattern
; that constitutes the head of all LISPLOG rules representing the system
; (with larger patterns, production-system usage can be parameterized).
; The deduction cycle of fone calls is controlled by backtracking, i.e.
; it proceeds in a single-step fashion governed by LISPLOG's more command.
; (n-solutions ... 1) avoids final cuts for all productions of all systems.
; A production of system s is notated by (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. A sample system like a below may be used by typing (fone (a)),
; followed by more, ... or typing (fall (a)); however, (fall (d)) diverges.
; For system c a trace with the spy command can be instructive.
```


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


```
; The forward-with-backward implementation:
(ass (fone _sy) (n-solutions _sy 1) (forward one _sy))  ; one step at a time
(ass (fall _sy) (not (forward all _sy)))                ; all steps together
(ass (forward one _sy))                                   ;reflexive  and
(ass (forward _x _sy) (n-solutions _sy 1) (forward _x _sy)) ;transitive closure
(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

; System c uses a conclusion containing an anonymous [ID] variable:
(ass (c) (ok ich) (ok du) (nap (roger ID)))        ; N=2 ['alles Roger!']
(ass (c) (sonntagskind _x) (nap (ok _x)))          ; N=1
(ass (sonntagskind du))                            ; 'working memory' fact 1
(ass (ok ich))                                     ; 'working memory' fact 2

; System d demonstrates an infinite transitive closure enumerable by fone:
(ass (d) (natural _x) (nap (natural (succ _x))))  ; N=1 [recursive production]
(ass (natural 0))                                  ; 'working memory' fact

; System e employs predicate parameters filled via (e parent brother uncle):
(ass (e _p _q _r) (_p _x _y) (_q _y _z) (nap (_r _x _z)))    ; N=2
(ass (parent nina gina))                           ; 'working memory' fact 1
(ass (brother gina tino))                          ; 'working memory' fact 2
```