The MVL Theorem Proving System

Matthew L. Ginsberg
Computer Science Department
Stanford University
Stanford, California 94305
ginsberg@cs.stanford.edu

The MVL theorem-proving system is an implementation of theoretical work currently under way at Stanford University. Theoretical progress made there is regularly included in MVL in order to ensure that the ideas developed theoretically can be tested experimentally as well.

The novel features of MVL are as follows:

1. The most important feature is the development of a theorem prover that uses generalized truth values, so that sentences can be labelled as defaults, assumptions, etc. in addition to the conventional values of true and false. The truth values must be chosen from a set known as a *bilattice*, but it is shown in [5] that this restriction is a very weak one. The system includes bilattices that reason in the following fashions:
   (a) First-order logic
   (b) Assumption-based truth maintenance as described in [2] and a first-order ATMS as described in [6]
   (c) Probabilistic reasoning
   (d) Default reasoning and hierarchical default reasoning
   (e) Circumscription, as discussed in [6]
   (f) Temporal reasoning, as discussed in [10]
2. Negative subgoals with free variables. Most knowledge representation systems are unable to deal with negative subgoals that contain unbound variables; as an example, consider the PROLOG fragment

   ```
   f :- not(g(x)).
   g(A).
   ```

   Given the query f, what are we to make of the floundered subgoal $\neg g(x)$?

   Two solutions to this have been proposed. One, due to David Chan and called *constructive negation* [1], suggests enumerating the solutions to the floundered subgoal and then interpreting the result. Ginsberg points out in [4] that there may be infinitely many such solutions and that useful information can often be obtained by instantiating the variables in the floundered subgoals to Skolem constants. MVL supports both of these ideas.
3. Modal operators. MVL has extensive features for dealing with modal operators, treating them as functions on the bilattice of underlying truth values. It is shown in [8] that this approach is a generalization of the possible-worlds construction due to Kripke [11]. Quantifiers are also treated in this fashion, and quantifying into modal expressions is permitted.
4. Anytime reasoning. The theoretical nature of the MVL theorem prover is such that it is generally possible to interrupt the inference process and obtain a useful partial answer before the reasoning system has completed its analysis. This is described in [5, 9] and supported by the implementation.
5. Control of inference. The first-order theorem prover underlying the MVL system incorporates checks to catch recursive inference paths as described in [12].
6. Unification. The system includes a fast unifier based on ideas appearing in [3]; the unifier includes an occurcheck and full support for sequence variables (variables that can match arbitrary sequences of object constants).

The MVL system is written in pure Common Lisp and has been ported to Symbolics machines, TI Explorers, Mac II's, Suns, NeXT machines and Hewlett-Packard workstations. Documentation is available [7]. The source code for the system (including documentation) is available via anonymous ftp from the MVL directory on t.stanford.edu.[1]

The remainder of this paper is an excerpted version of the MVL test suite. The output has been changed in some insignificant ways in order to make the presentation somewhat clearer. User commands are preceded by ->.

```
-> (load-logic)          ; select a new bilattice.
                         ; The system displays the
                         ; available choices:

  (a)  ATMS
  (b)  ATMS/CNF
  (c)  Circumscription
  (d)  Default
  (f)  First-order
  (h)  Hierarchical default
  (n)  Nonmonotonic ATMS
  (o)  First-order ATMS
  (p)  Probability
  (t)  Temporal

Select a logic by letter: First-order logic.
"First-order logic."     ; first-order logic is
                         ; selected by the user

-> (mvl-load "f1")       ; a declarative database is
                         ; loaded that contains a
                         ; description of a small
                         ; digital circuit (a full
                         ; adder)
DONE

-> (bc '(val (out 1 f1) 1 ?x))
                         ; what is the value of
                         ; output 1 of circuit f1 at
                         ; time 1?  bc invokes the
                         ; backward chainer and ?x
                         ; is a variable
```

[1] Please do not retrieve the system during business hours Pacific time, since the machine is frequently being used quite heavily at that time.

```
#S(ANSWER :BINDING ((?X . OFF))
          :VALUE TRUE)
                        ; The answer returned binds
                        ; ?x to OFF and the truth
                        ; value of TRUE indicates
                        ; certainty

-> (state '(<= (a) (b)))
                        ; here, the user enters a
                        ; few facts designed to
                        ; check the control of
                        ; recursive inference.  This
                        ; fact says that (b) entails
                        ; (a).
(P61)                   ; the system labels the
                        ; facts with consecutive
                        ; proposition numbers

-> (state '(<= (b) (c) (b)))
                        ; here is the loop -- the
                        ; PROLOG equivalent is
                        ; B :- C,B.
(P62)

-> (state '(c))
(P63)

-> (bc '(a))            ; can you derive A given
                        ; this information?
NIL                     ; the loop is detected and
                        ; failure is returned

-> (load-logic #\a)     ; here we switch to a new
                        ; logic --
"ATMS."                 ; assumption-based truth
                        ; maintenance

-> (mvl-load "f1")      ; the circuit description is
                        ; loaded again.  Note that
                        ; the identical file is used
DONE

-> (bc '(val (out 1 f1) 1 ?x))
                        ; what can you say about
                        ; output 1 now?
#S(ANSWER :BINDING ((?X . OFF))
          :VALUE (((P68 P67))))
                        ; it's still off, but now
                        ; the truth value reflects
                        ; the fact that two
                        ; assumptions were used to
                        ; derive it

-> (denotes 'p68)       ; what is the first
                        ;assumption?
(TYPE X2 XORG)          ; that X2 is a (functioning)
                        ; XOR gate

-> (denotes 'p67)       ; the second assumption is
(TYPE X1 XORG)          ; that X1 is an XOR gate.
                        ; The form of the
                        ; assumptions is relevant to
                        ; diagnostic applications

-> (load-logic #\d)     ; switch to default logic
"Default logic."

-> (mvl-load "def-test")
                        ; load some stuff about
                        ; birds and flying
DONE

-> (contents)           ; what's in the database?
(<= (FLIES ?X) (BIRD ?X)) has truth value DT
(<= (NOT (FLIES ?X)) (OSTRICH ?X)) has truth value T
(<= (BIRD ?X) (OSTRICH ?X)) has truth value T
(<= (NOT (ACROPHOBIC ?X)) (FLIES ?X))
   has truth value T
(OSTRICH FRED) has truth value T
(BIRD TWEETY) has truth value T

;; The following facts:
;;  Birds fly (true by default only)
;;  Ostriches do not fly
;;  Ostriches are birds
;;  Flying things are not acrophobic
;;  Fred is an ostrich
;;  Tweety is a bird

-> (bcs '(not (acrophobic ?x)))
                        ; who is not acrophobic?
                        ; The plural version bcs
                        ; returns all answers, not
                        ; just the first one
(#S(ANSWER :BINDING ((?X . TWEETY))
           :VALUE DT))
                        ; Tweety is the only answer
                        ; found and the proof is
                        ; only by default

-> (load-logic #\c)     ; switch to circumscription
"Circumscription."

-> (mvl-load "circ-test")
DONE                    ; load the test data

-> (contents)           ; what's in the database?
(<= (BLOCK A) (NOT (BLOCK B))) has truth value T
(BLOCK ?X) has truth value D(NIL ((P110)))

;; A sentence saying that A is a block or B is, and
;; another sentence saying that by default, nothing
;; is a block.  In other words, the circumscription
;; should minimize the extent of the BLOCK
;; predicate.  The leading "D" in the truth value
;; indicates that it is a default.

;; The query is whether you can show that A and B
;; aren't *both* blocks.  This should succeed, since
;; there are two minimal models.  In one A is the
;; only block; in the other, B is.  Therefore they
;; can't *both* be blocks.  This example is
;; McCarthy's.

-> (circum '(not (and (block a) (block b))))

Invoking multivalued prover on
   (NOT (AND (BLOCK A) (BLOCK B))).
                        ; first the system sees what
                        ; assumptions it needs to
                        ; prove the query
Values returned are:
   binding lists: (NIL)
   truth values:  (D((((P110 (?X . B)))
                       ((P110 (?X . A))))))
                        ; it needs to use the
```

```
                          ; default rule, applied
                          ; *either* to A or to B

Checking circumscriptive context for truth value
   D((((P110 (?X . B))) ((P110 (?X . A))))).
                          ; the consistency check
Checking consistency for
   (OR (NOT (BLOCK B)) (NOT (BLOCK A))).

#S(ANSWER :BINDING NIL  ; the consistency check
          :VALUE T)     ; succeeds and the sentence
                          ; therefore follows from the
                          ; circumscription axiom

-> (load-logic #\h)     ; switch to hierarchical
                          ; default logic to check
                          ; completed database
"Hierarchical default logic."

-> (mvl-load "completed-db")
DONE                      ; test data

-> (contents)           ; what's in the database?
(<= (F ?X) (NOT (G ?X))) has truth value T
(<= (G ?X) (NOT (H ?X))) has truth value T
(H A) has truth value T
(G ?X) has truth value D(2)F
(H ?X) has truth value DF

;; The last two axioms say that by default nothing
;; is an H, and nothing is a G by a second-order
;; default.  The first axioms say that F will be
;; true for X if G is false for X, and that G will
;; be true if H is false.  H is true for A.

-> (bc '(f ?x))         ; for what X is f true?

;; The system reasons as follows:  First, it
;; realizes that it can prove f(x) by assuming
;; not g(x).  A consistency check requires that it
;; attempt to prove g(x), and this succeeds using
;; the default assumption not h(x); the original
;; query therefore fails.  (The default assumption
;; about h takes precedence because the default
;; assumption about G is second-order.)  But since
;; the default assumption not h(x) actually *fails*
;; if x is equal to A, it is worth trying again with
;; x bound to A.  This succeeds, and the prover
;; finally is able to return A as an answer to the
;; original query.  The following diagnostics were
;; output by the program:

Prover for (F ?X) invoked.
Returns binding NIL using assumption ((NOT (G ?X))).
Prover for (G ?X) invoked.
Returns binding NIL using assumption ((NOT (H ?X))).
Prover for (H ?X) invoked.
Returns binding ((?X . A)) and assumption list NIL.
 ... now attempting to prove (G A).
Prover for (G A) invoked.
Prover for (G A) terminated.

#S(ANSWER :BINDING ((?X . A))
          :VALUE D(2)T)

-> (load-logic #\f)     ; back to first-order logic
"First-order logic."    ; to test modal operators
```

```
-> (state '(a))         ; you know A
(P118)
-> (state '(<= (not (b)) (a)))
(P119)                    ; and that A implies not B

-> (bc '(l (not (b))))  ; do you know not B?  (The
                          ; modal operator L denotes
                          ; "I know that")
#S(ANSWER :BINDING NIL  ; yes!
          :VALUE TRUE)

-> (bcs '(m (b)))       ; is B possible?  The modal
                          ; operator M denotes "it is
                          ; possible that"
NIL                       ; no, B is not possible

-> (state '(f a))       ; a simple example involving
(P120)                    ; quantification.  You know
                          ; f(a) and g(a)

-> (state '(g a))
(P121)

-> (bc '(forall ?x (<= (g ?x) (L (f ?x)))))
                          ; is it the case that for
                          ; every x, if you know f(x),
                          ; then g(x) holds?
#S(ANSWER :BINDING NIL  ; yes!
          :VALUE TRUE)

-> (state '(f b))       ; how about if you also know
(P122)                    ; f(b)?

-> (bc '(forall ?x (<= (g ?x) (L (f ?x)))))
NIL                       ; now the query fails, since
                          ; we know f(b) but cannot
                          ; prove g(b)

;; The final example involves reasoning about action
;; in a domain where there is a gun and an
;; individual named Fred.  Initially, the gun is
;; loaded and Fred is alive.  At time 1, the gun is
;; unloaded; at time 2, the shooting is attempted
;; (it shouldn't work, though) and the gun is
;; reloaded.  At  time 50, the shooting is attempted
;; again.  It should work this time,  since the gun
;; has presumably remained loaded between times 3
;; and 50.  Just to make sure, though, the gun is
;; reloaded at time 52 and the  shooting is tried
;; again at time 53.  The action of loading or
;; unloading the gun takes a single unit of time to
;; be effective; shooting Fred takes two units of
;; time.

;; This example shows the ability of the system to
;; interrupt its inference activities and return
;; partial answers.  The program will be asked,
;; "What can you say about Fred's being alive?"

-> (load-logic #\t)     ; logic for temporal
"Temporal reasoning."   ; reasoning

-> (mvl-load "actions") ; declarative description
DONE                      ; of actions generally

-> (mvl-load "shooting")
DONE                      ; declarative description of
                          ; this domain
```

```
-> (bc '(idn (holds alive)))
                              ; what about Fred, then?

Answer found:
  time 52, value DF          ; at time 52, he is
                              ; presumably dead, assuming
                              ; that the gun remained
                              ; loaded

Answer found:
  time 4, value DF           ; at time 4, he is
                              ; presumably dead, assuming
                              ; that the gun remained
                              ; loaded from the initial
                              ; situation to time 2
  time 52, value DF          ; at time 52, the assumption
                              ; is that the gun remained
                              ; loaded from the load
                              ; action at time 2 to time
                              ; 50

Answer found:
  time 52, value DF          ; oops!  The first shooting
                              ; fails because the gun was
                              ; unloaded at time 1

Answer found:
  time 52, value DF
  time 55, value F           ; the final shooting is
                              ; guaranteed to succeed
                              ; because the gun was just
                              ; loaded

Answer found:
  time 0, value DT           ; presumably, he'll survive
                              ; until someone kills him
  time 52, value DF          ; ... which they probably do
                              ; at time 52
  time 55, value F

Answer found:
  time 0, value T            ; in fact, he's *known* to
                              ; be alive in the initial
                              ; situation
  time 1, value DT           ; so the presumption of life
                              ; is only at times 1 and
                              ; later
  time 52, value DF
  time 55, value F

Answer complete.
```

Acknowledgement

The development of MVL has been supported by Rockwell International, General Dynamics, the National Science Foundation, the Air Force Office of Sceintific Research and the Defense Advanced Research Projects Agency.

References

[1] D. Chan. Constructive negation based on the completed database. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 111–125, 1988.

[2] J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[3] G. Escalada-Imax and M. Ghallab. A practically efficient and almost linear unification algorithm. *Artificial Intelligence*, 36:249–263, 1988.

[4] M. L. Ginsberg. Negative subgoals with free variables. *Journal of Logic Programming.* To appear.

[5] M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[6] M. L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[7] M. L. Ginsberg. User's guide to the MVL system. Technical report, Stanford University, 1989.

[8] M. L. Ginsberg. Bilattices and modal operators. *Journal of Logic and Computation*, 1, 1990.

[9] M. L. Ginsberg. Anytime declarativism. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1991. Submitted.

[10] M. L. Ginsberg. Computational considerations in reasoning about action. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1991.

[11] S. A. Kripke. Semantical considerations on modal logic. In L. Linsky, editor, *Reference and Modality*, pages 63–72. Oxford University Press, London, 1971.

[12] D. E. Smith, M. R. Genesereth, and M. L. Ginsberg. Controlling recursive inference. *Artificial Intelligence*, 30:343–389, 1986.