SNePSLOG User's Manual

by

Donald P. McKay and Joao Martins

SNERG
Technical Note No. 4
Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

May 1981

SNePSLOG is a logic programming interface to SNePS. It uses SNalculus, the basic predicate calculus augmented with SNePS logical connectives. The system consists of two ATN grammars, one for parsing and one for generation. In addition, there is a toplevel READ-EVAL-PRINT loop which takes the place of the SNePS toplevel. The SNePSLOG language is described by the context free grammar in Appendix I. If you have any comments or suggestions, please write them down and direct them to one of the authors.

To run SNePSLOG, use (LOAD '(SNLOG CSDLIB)), enter SNePS and call the function SNEPSLOG. The function SNEPSLOG takes a number of optional keyword arguments to control the kind of output generated. Such arguments are specified by typing in a keyword followed by its value. The possible keywords are:

1. LOGTRACE: Controls the amount of information displayed about SNEPSUL expressions. It has two possible values: USER and LOGIC-HACKER. The default is user which causes the output to consist only of the result of the SNePS action and the

running time. The value LOGIC-HACKER generates a more detailed output. Besides the values printed out by the USER option the system also prints the SNePSUL expression equivalent to the SNePSLOG input, its SNePS value and the parsing and generation times.

2. TLVLXX: Controls the parser and generator traces. Including TLVLXX <some trace level> sets the ATN's trace level. Defaults to -1.
3. INFERTRACE: Controls the tracing of the inference system. Defaults to SURFACE.

For example, if the user wants to see the SNePSUL expression generated by his input and also wants to follow the workings of the ATNs grammars with trace level 4 he should call SNEPSLOG as:

```
(SNEPSLOG LOGTRACE LOGIC-HACKER TLVLXX 4)
```

See the examples in Appendix II. To exit SNePSLOG type END. Note, SNePSLOG distinguishes lower case and upper case letters so be careful. For example, universal quantification is indicated with the symbols "ALL", not "All" or "all". Also, the individual symbols in the grammar of Appendix I are reserved symbols. For example, any occurrence of "V" is treated as the connective for OR.

Suggested improvements

It is painfully obvious that adding a new front end facility will further shrink the available memory for use in the SNePS network. What we have found so far is that it is impossible to run the parts of the SNePS system required in interpreted form interactively. With a compiled system, the system will be usable for very small to small examples. This is extremely unfortunate because SNePSLOG is a syntax which should be explored further, possibly not on the CYBER. One programming effort that could reduce the size of the system and increase the efficiency of the parsing is to recode the parsing ATN directly in LISP functions. Another possibility is to leave out as much of ALISP as possible, such as the editor, etc. One way to get rid of the reserved symbol "V" is to just allow SNePS connectives to be used, i.e. ban "V", "&" and "<=>".

Appendix I - The SNePSLOG grammar

A WARNING - as of today (4/20/81), the parsing grammar has no error messages. Hopefully, someone will take it upon themselves to extend this basic facility into a more friendly user interface.

The language accepted by the SNePSLOG function is the language accepted by the following CFG:

```
    notational conventions:
     - nonterminal symbols appear in angle brackets
     - terminal symbols appear between " marks
     - standard grammar metasymbols
         ¦ for alternation
        [] enclose optional parts
         * Kleene star (zero or more repetitions)
         + Kleene plus (one or more repetitions)
        {} enclose comments or semantic interpretation
        () group alternatives


 <command> -> [<com letter>] <wff> ¦ <wff> [<com letter>]
              ¦ <wff> (assumed ".")

 <com letter> -> "." ¦ "!" ¦ "?" ¦ "??"
              {. is BUILD
               ! is ADD
               ?? is FIND
               ? is DEDUCE}

 <wff> -> <ent-wff> [ <=> <ent-wff> ]


 <ent-wff> -> <wff'> ¦ (<wff'> ¦ <balanced list of wffs>)
                  <entailment>
                  (<wff> ¦ <balanced list of wffs>)


 <wff'> -> <wff''> ¦ <wff''> ["V" <wff''>]+

 <wff''> -> <wff'''> ¦ <wff'''> ["&" <wff'''>]+


 <wff'''> -> <leftparen> <wff> <rightparen> ¦
             "~" <wff> ¦
             "DELTA" <leftparen> <wff> <rightparen> ¦
```

```
            "¦-/-" <wff> ¦
            "^" <atomic formula> (the predicate is a
                                  function node) ¦
            "ANDOR" <leftparen> <min> <max> <rightparen>
                    <balanced list of wffs> ¦
            "THRESH" <leftparen> <threshold> <rightparen>
                     <balanced list of wffs> ¦
            "ALL" <variable list> <wff> ¦
            "EXISTS" <variable list> <wff> ¦
            "NEXISTS" <number list><variable list><leftparen>
                      <wff>+ ":" <wff>+ <rightparen> ¦
            <atomic formula>

<balanced list of wffs> -> <leftparen> <wff>+ <rightparen>

<entailment> -> &=> ¦ V=> ¦ i=> (for some integer i)

<variable list> -> <leftparen> <atom>+ <rightparen>
                       (where <atom> is a LISP atom)

<number list> -> <leftparen> (<number> ¦ "_") <rightparen>
                 (There are at most three numbers and they
                  are interpreted as the minimum, the maximum
                  and the total respectively. "_" indicates an
                  omitted value.)

<atomic formula> -> [("%" ¦ "*")] <atom> [<argument list>]
                  (If "%" or "*" precedes <atom> then they
                   are interpreted as SNePS macro characters.
                   "%" creates a new temporary variable and
                   "*" references the value of a SNePSUL
                   variable).

<argument list> -> <leftparen> <atomic formula>+ <rightparen>


<leftparen> -> "(" ¦ "[" ¦ "{"

<rightparen> -> ")" ¦ "]" ¦ "}"
```

Appendix II - Sample sessions

SESSION 1

```
? (LOAD '(SNLOG CSDLIB))
SNLOG
? (SNEPS)
SNEPS
** (DEFINE R R- A1 A1- A2 A2- A3 A3- A4 A4- A5 A5- A6 A6-)
(R R-)
(A1 A1-)
(A2 A2-)
(A3 A3-)
(A4 A4-)
(A5 A5-)
(A6 A6-)
(DEFINED)
45 MSECS

** (^(SNEPSLOG))
Entering READLINE parse loop, type END to stop
(Input need not be enclosed in parenthesis)
>  ;
>  ; Ships in the Converted Forrest Sherman Class have weapons
>  ; of type CFS-W and sensors of type CFS-S.
>  ;
>  ALL(S)[ Class(S,Converted-Forrest-Sherman)
>          V=>
>          {Weapons(S,CFS-W), Sensors(S,CFS-S)}]

> Surface description of value:
ALL(S)[Class(S,Converted-Forrest-Sherman)
       V=>
       {Sensors(S,CFS-S),Weapons(S,CFS-W)}]
6282  Msecs

>  ;
>  ; The weapons of type CFS-W are one single Tartar, one
>  ; Mk-42, one ASROC 8-tube and two Mk-32.
>  ;
>  ALL(S)[ Weapons(S,CFS-W)
>          V=>
>          { Has(S,one,Single-Tartar),
>            Has(S,one,Mk-42),
>            Has(S,one,ASROC-8-tube),
>            Has(S,two,Mk-32) }]

> Surface description of value:
ALL(S)[Weapons(S,CFS-W)
       V=>
       {Has(S,two,Mk-32),Has(S,one,ASROC-8-tube),
        Has(S,one,Mk-42),Has(S,one,Single-Tartar)}]
```

```
13515  Msecs


>  ;
>  ; The sensors of type CFS-S are one SQS-23, one SPS-10,
>  ; either one of SPS-37 or SPS-40 and one SPS-48.
>  ;
>  ALL(S)[ Sensors(S,CFS-S)
>          V=>
>          ( Has(S,one,SQS-23),
>            Has(S,one,SPS-10),
>            ANDOR(1,1){ Has(S,one,SPS-37),
>                        Has(S,one,SPS-40) },
>            Has(S,one,SPS-48) }]

> Surface description of value:
ALL(S)[Sensors(S,CFS-S)
       V=>
       (Has(S,one,SPS-48),
        ANDOR(1,1)(Has(S,one,SPS-40),Has(S,one,SPS-37)},
        Has(S,one,SPS-10),Has(S,one,SQS-23)}]
16512  Msecs

>  ;
>  ; By default, the sensors of type CFS-S have one SPS-37
>  ;
>  ALL(S)[ Sensors(S,CFS-S) V=>  DELTA(Has(S,one,SPS-37)) ]

> Surface description of value:
ALL(S)[Sensors(S,CFS-S) V=>  (DELTA(Has(S,one,SPS-37))}]
2369  Msecs


>  ;
>  ; Does Decatur have one SPS-10?
>  ;
>  Has(Decatur,one,SPS-10) ?
>

Surface description of value:


1394  Msecs

>  ;
>  ; Decatur is of class Converted-Forrest-Sherman!
>  ;
>  Class(Decatur,Converted-Forrest-Sherman) !
>

SINCE
Class(Decatur,Converted-Forrest-Sherman)


WE INFER
Sensors(Decatur,CFS-S)
```

```
SINCE
Sensors(Decatur,CFS-S)


WE INFER
Has(Decatur,one,SPS-10)



Surface description of value:
Class(Decatur,Converted-Forrest-Sherman)
Sensors(Decatur,CFS-S)
Has(Decatur,one,SPS-10)
3372  Msecs

>  ;
>  ; John-Paul-Jones is of class
>  ; Converted-Forrest-Sherman!
>  ;
>  Class(John-Paul-Jones,Converted-Forrest-Sherman) !
>
SINCE
Class(John-Paul-Jones,Converted-Forrest-Sherman)


WE INFER
Weapons(John-Paul-Jones,CFS-W) Sensors(John-Paul-Jones,CFS-S)


SINCE
Weapons(John-Paul-Jones,CFS-W)


WE INFER
Has(John-Paul-Jones,one,Single-Tartar)
Has(John-Paul-Jones,one,Mk-42)
Has(John-Paul-Jones,one,ASROC-8-tube)
Has(John-Paul-Jones,two,Mk-32)


SINCE
Sensors(John-Paul-Jones,CFS-S)


WE INFER
DELTA(Has(John-Paul-Jones,one,SPS-37))


SINCE
Sensors(John-Paul-Jones,CFS-S)


WE INFER
Has(John-Paul-Jones,one,SQS-23)
Has(John-Paul-Jones,one,SPS-10)
ANDOR(1,1)(Has(John-Paul-Jones,one,SPS-37),
            Has(John-Paul-Jones,one,SPS-40))
Has(John-Paul-Jones,one,SPS-48)


SINCE
Has(John-Paul-Jones,one,SPS-37)


WE INFER
~Has(John-Paul-Jones,one,SPS-40)



Surface description of value:
Class(John-Paul-Jones,Converted-Forrest-Sherman)
Weapons(John-Paul-Jones,CFS-W)
Sensors(John-Paul-Jones,CFS-S)
Has(John-Paul-Jones,one,Single-Tartar)
Has(John-Paul-Jones,one,Mk-42)
Has(John-Paul-Jones,one,ASROC-8-tube)
Has(John-Paul-Jones,two,Mk-32)
Has(John-Paul-Jones,one,SQS-23)
Has(John-Paul-Jones,one,SPS-10)
Has(John-Paul-Jones,one,SPS-48)
Has(John-Paul-Jones,one,SPS-37)
~Has(John-Paul-Jones,one,SPS-40)
23448  Msecs

>  END
> (SNePSLOG)
```

SESSION 2

```
? (LOAD '(SNLOG CSDLIB))
SNLOG
? (SNEPS)
SNEPS
** (DEFINE R R- A1 A1- A2 A2- A3 A3- A4 A4- A5 A5- A6 A6-)

(R R-)
(A1 A1-)
(A2 A2-)
(A3 A3-)
(A4 A4-)
(A5 A5-)
(A6 A6-)
(DEFINED)
45 MSECS

** (^(SNEPSLOG LOGTRACE LOGIC-HACKER))
Entering READLINE parse loop, type END to stop
```

(Input need not be enclosed in parenthesis)

```
 >  ALL (x) Hard-worker(x) V=> CS-Major(x) V=> Programmer(x)
 > SNePSUL expression:  (FORBTOP
                         AVB
                         (($ 'x))
                         ANT
                         (FORBNOTOP R Hard-worker A1 (* 'x))
                         CQ
                         (FORBNOTOP ANT
                                    (FORBNOTOP R CS-Major
                                               A1 (* 'x))
                                    CQ
                                    (FORBNOTOP R Programmer
                                               A1 (* 'x))))
 Parse time: 6697  Msecs
 Value of expression:  ((M9))
 Evaluation time:  572  Msecs
 Surface description of value:
 ALL(x)[Hard-worker(x) V=> (CS-Major(x) V=> Programmer(x))]

 Generation time:  3574  Msecs

 >  ALL (x) Clever(x) V=> CS-Major(x) V=> Programmer(x)
 > SNePSUL expression:  (FORBTOP
                         AVB
                         (($ 'x))
                         ANT
                         (FORBNOTOP R Clever A1 (* 'x))
                         CQ
                         (FORBNOTOP ANT
                                    (FORBNOTOP R CS-Major
                                               A1 (* 'x))
                                    CQ
                                    (FORBNOTOP R Programmer
                                               A1 (* 'x))))
 Parse time: 5793  Msecs
 Value of expression:  ((M18))
 Evaluation time:  1660  Msecs
 Surface description of value:
 ALL(x)[Clever(x) V=> (CS-Major(x) V=> Programmer(x))]

 Generation time:  2655  Msecs

 >  ALL (x) [Programmer(x), Knows(x,Lisp)] &=> AI-Bum(x)
 > SNePSUL expression:  (FORBTOP
                         AVB
                         (($ 'x))
                         &ANT
                         (FORBNOTOP R Programmer A1 (* 'x))
                         &ANT
                         (FORBNOTOP R Knows A1 (* 'x) A2 (Lisp))
                         CQ
                         (FORBNOTOP R AI-Bum A1 (* 'x)))
 Parse time: 10356  Msecs
 Value of expression:  ((M25))
 Evaluation time:  408  Msecs
```

```
Surface description of value:
ALL(x)[{Knows(x,Lisp),Programmer(x)} &=> AI-Bum(x)]

Generation time:  3507  Msecs

>  ALL (x) Programmer(x) V=> Computer-Bum(x)
> SNePSUL expression:  (FORBTOP AVB
                                (($ 'x))
                                ANT
                                (FORBNOTOP R Programmer
                                           A1 (* 'x))
                                CQ
                                (FORBNOTOP R Computer-Bum
                                           A1 (* 'x)))
Parse time: 4292  Msecs
Value of expression:  ((M30))
Evaluation time:  275  Msecs
Surface description of value:
ALL(x)[Programmer(x) V=> Computer-Bum(x)]

Generation time:  2566  Msecs

>  Clever(Joe)
> SNePSUL expression:  (FORBTOP R Clever A1 (Joe))
Parse time: 1159  Msecs
Value of expression:  ((M31))
Evaluation time:  42  Msecs
Surface description of value:
Clever(Joe)

Generation time:  470  Msecs

>  Knows(Joe,Lisp).
> SNePSUL expression:  (FORBTOP R Knows A1 (Joe) A2 (Lisp))
Parse time: 2540  Msecs
Value of expression:  ((M32))
Evaluation time:  66  Msecs
Surface description of value:
Knows(Joe,Lisp)

Generation time:  731  Msecs

>  Hard-worker(Joe)!
> SNePSUL expression:  (ADD R Hard-worker A1 (Joe))
Parse time: 1258  Msecs

SINCE
Hard-worker(Joe)

WE INFER
(CS-Major(Joe) V=> Programmer(Joe))


Value of expression:  (M33)
Evaluation time:  3953  Msecs
Surface description of value:
Hard-worker(Joe)
```

```
Generation time:  1470  Msecs

>  AI-Bum(Joe)?
> SNePSUL expression:  (DEDUCE R AI-Bum A1 (Joe))
Parse time: 1291  Msecs

SINCE
Clever(Joe)

WE INFER
(CS-Major(Joe) V=> Programmer(Joe))


Value of expression:  NIL
Evaluation time:  5187  Msecs
Surface description of value:

Generation time:  17  Msecs

>  !CS-Major(Joe)
> SNePSUL expression:  (ADD R CS-Major A1 (Joe))
Parse time: 2202  Msecs

SINCE
CS-Major(Joe)

WE INFER
Programmer(Joe)

SINCE
Programmer(Joe) Knows(Joe,Lisp)


WE INFER
AI-Bum(Joe)

SINCE
CS-Major(Joe)

WE INFER
Programmer(Joe)


Value of expression:  (M34 M35 M36)
Evaluation time:  9135  Msecs
Surface description of value:
CS-Major(Joe) Programmer(Joe) AI-Bum(Joe)

Generation time:  2311  Msecs

>  END
> (SNePSLOG)
```