Knowledge Representation and Reasoning
Logics for Artificial Intelligence

Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000

shapiro@cse.buffalo.edu

Contents

Part I

Part II

Part III

8 SNePS

8.1 SNePSLOG Semantics
The Intensional Domain
of (Mental) Entities

Frege: *The Morning Star is the Evening Star.*
different from *The Morning Star is the Morning Star.*

Russell: *George IV wanted to know whether Scott was the author of Waverly.*
not *George IV wanted to know whether the author of Waverly was the author of Waverly.*

Jerry Siegel and Joe Shuster: Clark Kent is a mild-mannered reporter; Superman is the man of steel.

Intensions vs. Extensions

the Morning Star and *the Evening Star*
Scott and *the author of Waverly*
Clark Kent and *Superman*

are different intensions, or intensional entities, or mental entities, or just entities,

even though they are coreferential, or extensionally equivalent, or have the same extensions.

SNePSLOG Semantics
Intensional Representation

SNePSLOG individual ground terms denote intensions, (mental) entities.

Mental entities include propositions.

Propositions are first-class members of the domain.

SNePSLOG wffs denote propositions.

Assume that for every entity in the domain there is a term that denotes it.

Make unique names assumption: no two terms denote the same entity.

The Knowledge Base

Think of the SNePS KB as the contents of the mind of an intelligent agent.

The terms in the KB denote mental entities that the agent has conceived of (so far).

Some of the wffs are **asserted**.
These denote propositions that the agent believes.

The rules of inference sanction believing some additional proposition(s), but drawing that inference is optional.
I.e., the agent is not logically omniscient.

8.2 SNePSLOG Syntax
Atomic Symbols

Individual Constants, Variables, Function Symbols:

any Lisp symbol, number, or string.

All that matters is the sequence of characters.
I.e. `"4"`, `\4`, and `4`, are the same.

The sets of individual constants, variables, and function symbols should be distinct, but don't have to be.

SNePSLOG Syntax
Terms

An individual constant is a term.

A variable is a term.

If `t1, ..., tn` are terms, then {`t1, ..., tn`} is a set of terms.

If `f` is a function symbol or a variable, then `f()` is a term.

If `t1, ..., tn` are terms or sets of terms and `f` is a function symbol or variable, then `f(t1, ..., tn)` is a term.

A function symbol needn't have a fixed arity, but it might be a mistake of formalization otherwise.

SNePSLOG Syntax
Atomic Wffs

If `x` is a variable, then `x` is a wff.
If `P` is a proposition-valued function symbol or variable,
then `P()` is a wff.

If `t1, ..., tn` are terms or sets of terms
and `P` is a proposition-valued function symbol or variable,
then `P(t1, ..., tn)` is a wff.

A predicate symbol needn't have a fixed arity, but it might be a mistake of formalization otherwise.

If P1, . . . , Pn are wffs, then {P1, . . . , Pn} is a set of wffs.

Abbreviation: If P is a wff, then P is an abbreviation of {P}.

Every wff is a proposition-denoting term.

SNePSLOG Syntax/Semantics
AndOr

If $\{P_1, \ldots, P_n\}$ is a set of wffs (proposition-denoting terms),
and i and j are integers such that $0 <= i <= j <= n$, then

`andor`$(i, j)\{P_1, \ldots, P_n\}$ is a wff (proposition-denoting term).

The proposition that at least i and at most j of $P_1, \ldots, P_n$ are True.

SNePSLOG Syntax/Semantics
Abbreviations of AndOr

$\texttt{\~{}}P = \texttt{andor(0,0)}\{P\}$

$\texttt{and}\{P_1, \ldots, P_n\} = \texttt{andor}(n, n)\{P_1, \ldots, P_n\}$
$\texttt{or}\{P_1, \ldots, P_n\} = \texttt{andor}(1, n)\{P_1, \ldots, P_n\}$
$\texttt{nand}\{P_1, \ldots, P_n\} = \texttt{andor}(0, n-1)\{P_1, \ldots, P_n\}$
$\texttt{nor}\{P_1, \ldots, P_n\} = \texttt{andor}(0, 0)\{P_1, \ldots, P_n\}$
$\texttt{xor}\{P_1, \ldots, P_n\} = \texttt{andor}(1, 1)\{P_1, \ldots, P_n\}$

P_1 `and` `...` `and` $P_n = \texttt{andor}(n, n)\{P_1, \ldots, P_n\}$
P_1 `or` `...` `or` $P_n = \texttt{andor}(1, n)\{P_1, \ldots, P_n\}$

SNePSLOG Syntax/Semantics
Thresh

If $\{P_1, \ldots, P_n\}$ is a set of wffs (proposition-denoting terms)
and i and j are integers such that $0 <= i <= j <= n$, then

`thresh`$(i, j)\{P_1, \ldots, P_n\}$ is a wff (proposition-denoting term).

The proposition that
either fewer than i or more than j of $P_1, \ldots, P_n$ are True.

SNePSLOG Syntax/Semantics
Abbreviations of Thresh

`iff`$\{P_1, \ldots, P_n\}$
is an abbreviation of `thresh`$(1, n-1)\{P_1, \ldots, P_n\}$

P_1 `<=>` $\cdots$ `<=>` P_n
is an abbreviation of `thresh`$(1, n-1)\{P_1, \ldots, P_n\}$

`thresh`$(i)\{P_1, \ldots, P_n\}$
is an abbreviation of `thresh`$(i, n-1)\{P_1, \ldots, P_n\}$

SNePSLOG Syntax/Semantics
Numerical Entailment

If $\{P_1, \ldots, P_n\}$ and $\{Q_1, \ldots, Q_m\}$ are sets of wffs (proposition-denoting terms), and i is an integer, $1 <= i <= n$, then
$\{P_1, \ldots, P_n\}$ i=> $\{Q_1, \ldots, Q_m\}$ is a wff (proposition-denoting term).

The proposition that whenever at least i of $P_1, \ldots, P_n$ are True, then so is any $Q_j \in \{Q_1, \ldots, Q_m\}$.

SNePSLOG Syntax/Semantics
Abbreviations of Numerical Entailment

$\{P_1, \ldots, P_n\}$ `=>` $\{Q1, \ldots, Qm\}$
is an abbreviation of $\{P_1, \ldots, P_n\}$ `1=>` $\{Q_1, \ldots, Q_m\}$

$\{P_1, \ldots, P_n\}$ `v=>` $\{Q1, \ldots, Qm\}$
is also an abbreviation of $\{P_1, \ldots, P_n\}$ `1=>` $\{Q_1, \ldots, Q_m\}$

$\{P_1, \ldots, P_n\}$ `&=>` $\{Q1, \ldots, Qm\}$
is an abbreviation of $\{P_1, \ldots, P_n\}$ n`=>` $\{Q_1, \ldots, Q_m\}$

SNePSLOG Syntax/Semantics
Universal Quantifier

If P is a wff (proposition-denoting term)
and $x_1, \ldots, x_n$ are variables, then

`all(`$x_1, \ldots, x_n$`)(`P`)` is a wff (proposition-denoting term).

The proposition that for every sequence of ground terms, $t_1, \ldots, t_n$,
$\mathrm{P}\{t_1/x_1, \ldots, t_n/x_n\}$ is True.

SNePSLOG Syntax/Semantics
Numerical Quantifier

If $\mathcal{P}$ and $\mathcal{Q}$ are sets of wffs, $x_1, \ldots, x_n$ are variables, and i, j, and k are integers such that $0 <= i <= j <= k$, then

`nexists`$(i, j, k)(x_1, \ldots, x_n)(\mathcal{P}:\quad \mathcal{Q})$ is a wff.

The proposition that there are k sequences of ground terms, $t_1, \ldots, t_n$, that satisfy every $P \in \mathcal{P}$, and, of them, at least i and at most j also satisfy every $Q \in \mathcal{Q}$.

SNePSLOG Syntax/Semantics
Abbreviations of Numerical Quantifier

`nexists(_, `j`, _)`$(x_1, \ldots, x_n)(\mathcal{P}:\quad \mathcal{Q})$
is an abbreviation of `nexists`$(0, j, \infty)(x_1, \ldots, x_n)(\mathcal{P}:\quad \mathcal{Q})$

`nexists(`i`, _, `k`)`$(x_1, \ldots, x_n)(\mathcal{P}:\quad \mathcal{Q})$
is an abbreviation of `nexists`$(i, k, k)(x_1, \ldots, x_n)(\mathcal{P}:\quad \mathcal{Q})$

SNePSLOG Syntax/Semantics
Wffs are Terms

Every wff is a proposition-denoting term.

So, e.g., `Believes(Tom, ~Penguin(Tweety))`
is a wff, and a well-formed term.

For a more complete, more formal syntax, see
The *SNePS 2.7.1 User's Manual,*
`http://www.cse.buffalo.edu/sneps/Manuals/manual271.pdf`.

8.3 SNePSLOG Proof Theory
Implemented Rules of Inference
Reduction Inference

Reduction Inference$_1$: If α is a set of terms and $\beta \subset \alpha$,

$\texttt{P(t}_1, \ldots, \alpha, \texttt{ ...t}_n\texttt{)} \vdash \texttt{P(t}_1, \ldots, \beta, \ldots \texttt{ t}_n\texttt{)}$

Reduction Inference$_2$: If α is a set of terms, and $\texttt{t} \in \alpha$,

$\texttt{P(t}_1, \ldots, \alpha, \texttt{ ...t}_n\texttt{)} \vdash \texttt{P(t}_1\texttt{, ..., t, ...t}_n\texttt{)}$

Example of Reduction Inference

```
: clearkb
Knowledge Base Cleared

: Member({Fido, Rover, Lassie}, {dog, pet}).
  wff1!:  Member({Lassie,Rover,Fido},{pet,dog})
 CPU time : 0.00

: Member ({Fido, Lassie}, dog)?
  wff2!:  Member({Lassie,Fido},dog)
 CPU time : 0.00
```

SNePSLOG Proof Theory
Implemented Rules of Inference
for AndOr

AndOr $\mathbf{I}_1$: $\mathtt{P}_1, \ldots, \mathtt{P}_n \vdash \mathtt{andor}(n,n)\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$

AndOr $\mathbf{I}_2$: $\mathtt{\sim P}_1, \ldots, \mathtt{\sim P}_n \vdash \mathtt{andor}(0,0)\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$

AndOr $\mathbf{E}_1$: $\mathtt{andor}(i,j)\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}, \mathtt{\sim P}_1, \ldots, \mathtt{\sim P}_{n-i} \vdash \mathtt{P}_j$
for $n - i < j \leq n$

AndOr $\mathbf{E}_2$: $\mathtt{andor}(i,j)\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}, \mathtt{P}_1, \ldots, \mathtt{P}_j \vdash \mathtt{\sim P}_k$,
for $j < k \leq n$

SNePSLOG Proof Theory
Implemented Rules of Inference
for Thresh

Thresh $\mathbf{E}_1$: When at least i args are true, and at least $n-j-1$ args are false, conclude that any other arg is true.

$\texttt{thresh}(i,j)\{\texttt{P}_1, \ldots, \texttt{P}_n\},$

$\texttt{P}_1, \ldots, \texttt{P}_i, \neg\texttt{P}_{i+1}, \ldots, \neg\texttt{P}_{i+n-j-1}$

$\vdash \texttt{P}_{i+n-j}$

Thresh $\mathbf{E}_2$: When at least $i-1$ args are true, and at least $n-j$ args are false, conclude that any other arg is false.

$\texttt{thresh}(i,j)\{\texttt{P}_1, \ldots, \texttt{P}_n\},$

$\texttt{P}_1, \ldots, \texttt{P}_{i-1}, \neg\texttt{P}_{i+1}, \ldots, \neg\texttt{P}_{i+n-j}$

$\vdash \neg\texttt{P}_i$

SNePSLOG Proof Theory
Implemented Rules of Inference
for &=>

&=>I: If $\mathcal{A}$, P$_1$, ..., P$_n$ $\vdash$ Q$_i$ for $1 \leq i \leq m$
then $\mathcal{A} \vdash$ {P$_1$, ..., P$_n$} &=> {Q$_1$, ..., Q$_m$}

&=>E: {P$_1$, ..., P$_n$} &=> {Q$_1$, ..., Q$_m$}, P$_1$, ..., P$_n$ $\vdash$ Q$_i$,
for $1 \leq i \leq m$

SNePSLOG Proof Theory
Implemented Rules of Inference
for v=>

v=>I: If $\mathcal{A} \vdash$ `P v=> Q` and $\mathcal{A} \vdash$ `Q v=> R` then $\mathcal{A} \vdash$ `P v=> R`

v=>E: $\{$`P`$_1$`,` $\ldots$`,` `P`$_n\}$ `v=>` $\{$`Q`$_1$`,` $\ldots$`,` `Q`$_m\}$`,` `P`$_i$`,`$\vdash$ `Q`$_j$,
for $1 \leq i \leq n, 1 \leq j \leq m$

SNePSLOG Proof Theory
Implemented Rules of Inference
for i=>

i=>E: $\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$ i=> $\{\mathtt{Q}_1, \ldots, \mathtt{Q}_m\}$, $\mathtt{P}_1, \ldots, \mathtt{P}_i \vdash \mathtt{Q}_j$,
for $1 \leq i \leq n, 1 \leq j \leq m$

SNePSLOG Proof Theory
Implemented Rules of Inference
for `all`

Universal Elimination for universally quantified versions of `andor`, `thresh`, `v=>`, `&=>`, and *i*`=>`.

UVBR & Symmetric Relations

In any substitution $\{t_1/x_1, \ldots, t_n/x_n\}$, if $x_i \neq x_j$, then $t_i \neq t_j$

```
: all(u,v,x,y)(childOf({u,v}, {x,y}) => Siblings({u,v})).

: childOf({Tom,Betty,John,Mary}, {Pat,Harry}).

: Siblings({?x,?y})?

  wff14!:  Siblings({Mary,John})
  wff13!:  Siblings({John,Betty})
  wff12!:  Siblings({Betty,Tom})
  wff11!:  Siblings({Mary,Betty})
  wff10!:  Siblings({John,Tom})
  wff9!:  Siblings({Mary,Tom})
```

SNePSLOG Proof Theory
Implemented Rules of Inference
for `nexists`

nexists $\mathbf{E}_1$:

$\mathtt{nexists(i,j,k)}(\overline{\mathtt{x}})(\overline{\mathtt{P}}(\overline{\mathtt{x}}) : \mathtt{Q}(\overline{\mathtt{x}})),$
$\overline{\mathtt{P}}(\overline{\mathtt{t}}_1), \mathtt{Q}(\overline{\mathtt{t}}_1), \ldots, \overline{\mathtt{P}}(\overline{\mathtt{t}}_\mathtt{j}), \mathrm{Q}(\overline{t}_j),$
$\overline{\mathtt{P}}(\overline{\mathtt{t}}_{\mathtt{j+1}})$
$\vdash \neg\mathtt{Q}(\overline{\mathtt{t}}_{\mathtt{j+1}})$

nexists $\mathbf{E}_2$:

$\mathtt{nexists(i,j,k)}(\overline{\mathtt{x}})(\overline{\mathtt{P}}(\overline{\mathtt{x}}) : \mathtt{Q}(\overline{\mathtt{x}})),$
$\overline{\mathtt{P}}(\overline{\mathtt{t}}_1), \neg\mathtt{Q}(\overline{\mathtt{t}}_1), \ldots, \overline{\mathtt{P}}(\overline{\mathtt{t}}_{\mathtt{k-i}}), \neg\mathtt{Q}(\overline{\mathtt{t}}_{\mathtt{k-i}}),$
$\overline{\mathtt{P}}(\overline{\mathtt{t}}_{\mathtt{k-i+1}})$
$\vdash \mathtt{Q}(\overline{\mathtt{t}}_{\mathtt{k-i+1}})$

8.4 Loading SNePSLOG

```
cl-user(2): :ld /projects/snwiz/bin/sneps
; Loading /projects/snwiz/bin/sneps.lisp
;;; Installing streamc patch, version 2.
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.7 [PL:2 2010/09/04 02:35:18] loaded.
Type ‘(sneps)’ or ‘(snepslog)’ to get started.

cl-user(3): (snepslog)



   Welcome to SNePSLOG (A logic interface to SNePS)

Copyright (C) 1984--2010 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type ‘copyright’ for detailed copyright information.
Type ‘demo’ for a list of example applications.
```

Running SNePSLOG

```
cl-user(3): (snepslog)

   Welcome to SNePSLOG (A logic interface to SNePS)

Copyright (C) 1984--2010 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type ‘copyright’ for detailed copyright information.
Type ‘demo’ for a list of example applications.

: clearkb
Knowledge Base Cleared
 CPU time : 0.00

: Member({Fido, Rover, Lassie}, {dog, pet}).
  wff1!:  Member({Lassie,Rover,Fido},{pet,dog})
 CPU time : 0.00

: Member ({Fido, Lassie}, dog)?
  wff2!:  Member({Lassie,Fido},dog)
 CPU time : 0.00
```

Common SNePSLOG Commands

```
: clearkb
Knowledge Base Cleared

: all(x)(dog(x) => animal(x)). ; Assert into the KB
  wff1!:  all(x)(dog(x) => animal(x))

: dog(Fido). ; Assert into the KB
  wff2!:  dog(Fido)

: dog(Fido)?? ; Query assertion without inference
  wff2!:  dog(Fido)
```

Common SNePSLOG Commands

```
: animal(Fido)?? ; Query assertion without inference

: animal(Fido)? ; Query assertion with inference
  wff3!:  animal(Fido)

: dog(Rover)! ; Assert into the KB & do forward inference
  wff6!:  animal(Rover)
  wff5!:  dog(Rover)

: list-asserted-wffs ; Print all asserted wffs
  wff6!:  animal(Rover)
  wff5!:  dog(Rover)
  wff3!:  animal(Fido)
  wff2!:  dog(Fido)
  wff1!:  all(x)(dog(x) => animal(x))
```

Tracing Inference

```
: trace inference
Tracing inference.

: animal(Fido)?

I wonder if  wff3:   animal(Fido)
holds within the BS defined by context default-defaultct

I wonder if  wff5:   dog(Fido)
holds within the BS defined by context default-defaultct

I know  wff2!:  dog({Rover,Fido})

Since  wff1!:  all(x)(dog(x) => animal(x))
and  wff5!:  dog(Fido)
I infer  wff3:   animal(Fido)

  wff3!:  animal(Fido)
 CPU time : 0.01

: untrace inference
Untracing inference.
 CPU time : 0.00

: animal(Rover)?
  wff6!:  animal(Rover)
```

Recursive Rules
Don't Cause Infinite Loops

```
: all(x,y)(parentOf(x,y) => ancestorOf(x,y)).
  wff1!:  all(y,x)(parentOf(x,y) => ancestorOf(x,y))

: all(x,y,z)({ancestorOf(x,y), ancestorOf(y,z)} &=> ancestorOf(x,z)).
  wff2!:  all(z,y,x)({ancestorOf(y,z),ancestorOf(x,y)} &=> {ancestorOf(x,z)})

: parentOf(Sam,Lou).
  wff3!:  parentOf(Sam,Lou)

: parentOf(Lou,Stu).
  wff4!:  parentOf(Lou,Stu)

: ancestorOf(Max,Stu).
  wff5!:  ancestorOf(Max,Stu)

: ancestorOf(?x,Stu)?
  wff8!:  ancestorOf(Sam,Stu)
  wff6!:  ancestorOf(Lou,Stu)
  wff5!:  ancestorOf(Max,Stu)
 CPU time : 0.01
```

Infinitely Growing Terms Get Cut Off

```
: all(x)(Duck(motherOf(x)) => Duck(x)).
  wff1!:  all(x)(Duck(motherOf(x)) => Duck(x))
 CPU time : 0.00

: Duck(Daffy)?

I wonder if  wff2:   Duck(Daffy)
holds within the BS defined by context default-defaultct

I wonder if  wff5:   Duck(motherOf(Daffy))
holds within the BS defined by context default-defaultct

I wonder if  wff8:   Duck(motherOf(motherOf(Daffy)))
holds within the BS defined by context default-defaultct
...
I wonder if  wff32:   Duck(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(Daffy))
holds within the BS defined by context default-defaultct
SNIP depth cutoff beyond *depthcutoffback* = 10

I wonder if  wff35:   Duck(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(mother(
holds within the BS defined by context default-defaultct
SNIP depth cutoff beyond *depthcutoffback* = 10
SNIP depth cutoff beyond *depthcutoffback* = 10
 CPU time : 0.05
```

Eager-Beaver Search

```
: all(x)(Duck(motherOf(x)) => Duck(x)).
  wff1!:  all(x)(Duck(motherOf(x)) => Duck(x))
: all(x)({walksLikeaDuck(x), talksLikeaDuck(x)} &=> Duck(x)).
  wff2!:  all(x)({talksLikeaDuck(x),walksLikeaDuck(x)} &=> {Duck(x)})
: and{talksLikeaDuck(Daffy),walksLikeaDuck(Daffy)}.
  wff5!:  walksLikeaDuck(Daffy) and talksLikeaDuck(Daffy)

: Duck(Daffy)? (1)
I wonder if  wff6:   Duck(Daffy)

I wonder if  wff9:   Duck(motherOf(Daffy))

I wonder if  wff3:   talksLikeaDuck(Daffy)

I wonder if  wff4:   walksLikeaDuck(Daffy)

It is the case that  wff4:   walksLikeaDuck(Daffy)

It is the case that  wff3:   talksLikeaDuck(Daffy)

Since  wff2!:  all(x)({talksLikeaDuck(x),walksLikeaDuck(x)} &=> {Duck(x)})
and  wff3!:  talksLikeaDuck(Daffy)
and  wff4!:  walksLikeaDuck(Daffy)
I infer  wff6:   Duck(Daffy)

  wff6!:  Duck(Daffy)
 CPU time : 0.02
```

Contradictions
The KB

```
: clearkb
Knowledge Base Cleared

: all(x)(nand{Mammal(x), Fish(x)}).
  wff1!:  all(x)(nand{Fish(x),Mammal(x)})

: all(x)(LivesInWater(x) => Fish(x)).
  wff2!:  all(x)(LivesInWater(x) => Fish(x))

: all(x)(BearsYoungAlive(x) => Mammal(x)).
  wff3!:  all(x)(BearsYoungAlive(x) => Mammal(x))

: LivesInWater(whale).
  wff4!:  LivesInWater(whale)

: BearsYoungAlive(whale).
  wff5!:  BearsYoungAlive(whale)
```

Contradictions
The Contradiction

```
: ?x(whale)?

  A contradiction was detected within context default-defaultct.
  The contradiction involves the newly derived proposition:
      wff6!:  Mammal(whale)

  and the previously existing proposition:
      wff7!:  ~Mammal(whale)

  You have the following options:
   1. [C]ontinue anyway, knowing that a contradiction is derivable;
   2. [R]e-start the exact same run in a different context which is
      not inconsistent;
   3. [D]rop the run altogether.

  (please type c, r or d)
=><= d
```

SNePSLOG Demonstrations

```
: demo
Available demonstrations:
  1:  Socrates - Is he mortal?
  2:  UVBR - Demonstrating the Unique Variable Binding Rule
  3:  The Jobs Puzzle - A solution with the Numerical Quantifie
  4:  Pegasus - Why winged horses lead to contradictions
  5:  Schubert's Steamroller
  6:  Rule Introduction - Various examples
  7:  Examples of various SNeRE constructs.
  8:  Enter a demo filename
Your choice (q to quit):
```

8.5 Reasoning Heuristics

Logically equivalent SNePSLOG wffs

are interpreted differently by the SNePS Reasoning System.

v=>-Elimination

Instead of

$$\frac{\texttt{P()} \qquad \texttt{(P() or Q())} \Rightarrow \texttt{R()}}{\texttt{R()}}$$

which would require or-I followed by =>-E

Have

$$\frac{\texttt{P()} \qquad \{\texttt{P(), Q()}\} \ \texttt{v=>}\ \texttt{R()}}{\texttt{R()}}$$

which requires only v=>-E

Example of v=>-E

```
: P().
  wff1!:  P()

: {P(), Q()} v=> R().
  wff4!:  {Q(),P()} v=> {R()}

: trace inference
Tracing inference.

: R()?
I wonder if  wff3:   R()
holds within the BS defined by context default-defaultct

I wonder if  wff2:   Q()
holds within the BS defined by context default-defaultct

I know  wff1!:  P()

Since  wff4!:  {Q(),P()} v=> {R()}
and  wff1!:  P()
I infer  wff3:   R()

  wff3!:  R()
```

Bi-Directional Inference
Backward Inference

```
: {p(), q()} v=> {r(), s()}.
  wff5!:  {q(),p()} v=> {s(),r()}

: p().
  wff1!:  p()

: r()?

I wonder if  wff3:   r()
holds within the BS defined by context default-defaultct

I wonder if  wff2:   q()
holds within the BS defined by context default-defaultct

I know  wff1!:  p()

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff3:   r()

  wff3!:  r()
```

Bi-Directional Inference
Forward Inference

```
: {p(), q()} v=> {r(), s()}.
  wff5!:  {q(),p()} v=> {s(),r()}

: p()!

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff4:   s()

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff3:   r()

  wff4!:  s()
  wff3!:  r()
  wff1!:  p()
```

Bi-Directional Inference
Forward-in-Backward Inference

```
: {p(), q()} v=> {r(), s()}.
  wff5!:  {q(),p()} v=> {s(),r()}

: r()?

I wonder if  wff3:   r()
holds within the BS defined by context default-defaultct

I wonder if  wff2:   q()
holds within the BS defined by context default-defaultct

I wonder if  wff1:   p()
holds within the BS defined by context default-defaultct

: p()!

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff3:   r()

  wff3!:  r()
  wff1!:  p()
```

Active connection graph cleared by `clear-infer`.

Bi-Directional Inference
Backward-in-Forward Inference

```
: p().
  wff1!:  p()

: p() => (q() => r()).
  wff5!:  p() => (q() => r())

: q()!

I know  wff1!:  p()

Since  wff5!:  p() => (q() => r())
and  wff1!:  p()
I infer  wff4:   q() => r()

I know  wff2!:  q()

Since  wff4!:  q() => r()
and  wff2!:  q()
I infer  wff3:   r()

  wff4!:  q() => r()
  wff3!:  r()
  wff2!:  q()
```

Modus Tollens Not Implemented

```
: all(x)(p(x) => q(x)).
  wff1!:  all(x)(p(x) => q(x))

: p(a).
  wff2!:  p(a)

: q(a)?
  wff3!:  q(a)

: ~q(b).
  wff6!:  ~q(b)

: p(b)?
:
```

Use Disjunctive Syllogism Instead

```
: all(x)(or{~p(x), q(x)}).
  wff1!:  all(x)(q(x) or ~p(x))

: p(a).
  wff2!:  p(a)

: q(a)?
  wff3!:  q(a)

: ~q(b).
  wff7!:  ~q(b)

: p(b)?
  wff9!:  ~p(b)
```

=> Is Not Material Implication

If $\Rightarrow$ is material implication,

$$\neg(P \Rightarrow Q) \Leftrightarrow (P \wedge \neg Q)$$

and

$$\neg(P \Rightarrow Q) \models P$$

But `~(p => q)` just means that its not the case that `p => q`:

```
: ~(p() => q()).
  wff4!:  ~(p() => q())

: p()?

:
```

Use or
Instead of Material Implication

```
: ~(~p() or q()).
  wff5!:  nor{q(),~p()}

: p()?

  wff1!:  p()
```

Ordering of Nested Rules Matters
Optimal Order

```
: wifeOf(Caren,Stu).
: wifeOf(Ruth,Mike).
: brotherOf(Stu,Judi).
: brotherOf(Mike,Lou).
: parentOf(Judi,Ken).
: parentOf(Lou,Stu).
: all(w,x)(wifeOf(w,x)
          => all(y)(brotherOf(x,y)
                    => all(z)(parentOf(y,z)
                              => auntOf(w,z)))).
: auntOf(Caren,Ken)?

I wonder if  wff8:   auntOf(Caren,Ken)
I wonder if  p7:   wifeOf(Caren,x)
I know  wff1!:  wifeOf(Caren,Stu)

I wonder if  p8:   brotherOf(Stu,y)
I know  wff3!:  brotherOf(Stu,Judi)

I wonder if  wff5!:  parentOf(Judi,Ken)
I know  wff5!:  parentOf(Judi,Ken)

  wff8!:  auntOf(Caren,Ken)
 CPU time : 0.03
```

Ordering of Nested Rules Matters

Bad Order

```
all(x,y)(brotherOf(x,y)
         => all(w)(wifeOf(w,x)
                   => all(z)(parentOf(y,z)
                             => auntOf(w,z)))).
: auntOf(Caren,Ken)?

I wonder if  wff8:   auntOf(Caren,Ken)
I wonder if  p1:   brotherOf(x,y)
I know  wff3!:  brotherOf(Stu,Judi)
I know  wff4!:  brotherOf(Mike,Lou)

I wonder if  wff12:   wifeOf(Caren,Mike)
I wonder if  wff1!:  wifeOf(Caren,Stu)
I know  wff1!:  wifeOf(Caren,Stu)

I wonder if  wff5!:  parentOf(Judi,Ken)
I know  wff5!:  parentOf(Judi,Ken)

  wff8!:  auntOf(Caren,Ken)
 CPU time : 0.04
```

Ordering of Nested Rules Matters Parallel

```
all(w,x,y,z)({wifeOf(w,x),brotherOf(x,y),parentOf(y,z)}
               &=> auntOf(w,z)).
: auntOf(Caren,Ken)?

I wonder if  wff8:   auntOf(Caren,Ken)
I wonder if  p5:   parentOf(y,Ken)
I wonder if  p2:   brotherOf(x,y)
I wonder if  p6:   wifeOf(Caren,x)

I know  wff5!:  parentOf(Judi,Ken)
I know  wff3!:  brotherOf(Stu,Judi)
I know  wff4!:  brotherOf(Mike,Lou)
I know  wff1!:  wifeOf(Caren,Stu)

  wff8!:  auntOf(Caren,Ken)
 CPU time : 0.03
```

Lemmas (Expertise)
Knowledge Base

```
: all(r)(transitive(r)
        => all(x,y,z)({r(x,y),r(y,z)} &=> r(x,z))).
: transitive(biggerThan).
: biggerThan(elephant,lion).
: biggerThan(lion,hyena).
: biggerThan(hyena,rat).
```

Lemmas: First Task

```
: biggerThan(?x,rat)?
I wonder if  p6:   biggerThan(x,rat)
I know  wff5!:  biggerThan(hyena,rat)
I wonder if  wff2!:  transitive(biggerThan)
I know  wff2!:  transitive(biggerThan)
I infer  wff6:   all(z,y,x)({biggerThan(x,y),biggerThan(y,z)} &=> {biggerThan(x,z)})
I wonder if  p8:   biggerThan(y,rat)
I wonder if  p10:   biggerThan(x,y)
I know  wff5!:  biggerThan(hyena,rat)
I wonder if  p12:   biggerThan(rat,z)
I know  wff3!:  biggerThan(elephant,lion)
I know  wff4!:  biggerThan(lion,hyena)
I infer  wff7:   biggerThan(lion,rat)
I infer  wff8:   biggerThan(elephant,rat)
...
  wff8!:  biggerThan(elephant,rat)
  wff7!:  biggerThan(lion,rat)
  wff5!:  biggerThan(hyena,rat)
 CPU time : 0.09
```

Second Task

```
: clear-infer
: biggerThan(truck,SUV).
: biggerThan(SUV,sedan).
: biggerThan(sedan,roadster).

: biggerThan(?x,roadster)?
I wonder if  p14:   biggerThan(x,roadster)
I know  wff11!:  biggerThan(sedan,roadster)
I wonder if  p10:   biggerThan(x,y)
I wonder if  p16:   biggerThan(y,roadster)
I know  wff3!:  biggerThan(elephant,lion)
I know  wff4!:  biggerThan(lion,hyena)
I know  wff5!:  biggerThan(hyena,rat)
I know  wff7!:  biggerThan(lion,rat)
I know  wff8!:  biggerThan(elephant,rat)
I know  wff9!:  biggerThan(truck,SUV)
I know  wff10!:  biggerThan(SUV,sedan)
I know  wff11!:  biggerThan(sedan,roadster)
I infer  wff12:   biggerThan(SUV,roadster)
I infer  wff13:   biggerThan(truck,roadster)
I wonder if  p17:   biggerThan(roadster,z)
  wff13!:  biggerThan(truck,roadster)
  wff12!:  biggerThan(SUV,roadster)
  wff11!:  biggerThan(sedan,roadster)
 CPU time : 0.04
```

Contexts

```
: demo /projects/shapiro/CSE563/Examples/SNePSLOG/facultyMeeting.snepslog
...
: ;;; Example of Contexts
;;; from
;;;   J. P. Martins & S. C. Shapiro, Reasoning in Multiple Belief Spaces IJCAI-83, 370-373.

: all(x)(meeting(x) => xor{time(x,morning), time(x,afternoon)}).
  wff1!:  all(x)(meeting(x) => (xor{time(x,afternoon),time(x,morning)}))
: all(x,y)({meeting(x),meeting(y)} &=> all(t)(xor{time(x,t),time(y,t)})).
  wff2!:  all(y,x)({meeting(y),meeting(x)} &=> {all(t)(xor{time(y,t),time(x,t)})})
: meeting(facultyMeeting).
  wff3!:  meeting(facultyMeeting)
: meeting(seminar).
  wff4!:  meeting(seminar)
: meeting(tennisGame).
  wff5!:  meeting(tennisGame)
: time(seminar,morning).
  wff6!:  time(seminar,morning)
: time(tennisGame,afternoon).
  wff7!:  time(tennisGame,afternoon)
: set-context stuSchedule {wff1,wff2,wff3,wff4,wff6}
((assertions (wff6 wff4 wff3 wff2 wff1)) (named (stuSchedule)) (kinconsistent nil))
: set-context tonySchedule {wff1,wff2,wff3,wff5,wff7}
((assertions (wff7 wff5 wff3 wff2 wff1)) (named (tonySchedule)) (kinconsistent nil))
: set-context patSchedule {wff1,wff2,wff3,wff4,wff5,wff6,wff7}
((assertions (wff7 wff6 wff5 wff4 wff3 wff2 wff1)) (named (patSchedule default-defaultct)) (kinconsistent nil))
```

Stu's Schedule

```
: set-default-context stuSchedule
((assertions (wff6 wff4 wff3 wff2 wff1)) (named (stuSchedule))
 (kinconsistent nil))

: list-asserted-wffs
  wff6!:  time(seminar,morning)
  wff4!:  meeting(seminar)
  wff3!:  meeting(facultyMeeting)
  wff2!:  all(y,x)({meeting(y),meeting(x)}
                   &=> {all(t)(xor{time(y,t),time(x,t)})})
  wff1!:  all(x)(meeting(x)
                 => (xor{time(x,afternoon),time(x,morning)}))

: time(facultyMeeting,?t)?
  wff10!:  time(facultyMeeting,afternoon)
  wff9!:  ~time(facultyMeeting,morning)
```

Tony's Schedule

```
: set-default-context tonySchedule
((assertions (wff7 wff5 wff3 wff2 wff1)) (named (tonySchedule))
 (kinconsistent nil))

: list-asserted-wffs
  wff12!:  xor{time(facultyMeeting,afternoon),time(facultyMeeting,morning)}
  wff7!:  time(tennisGame,afternoon)
  wff5!:  meeting(tennisGame)
  wff3!:  meeting(facultyMeeting)
  wff2!:  all(y,x)({meeting(y),meeting(x)}
                   &=> {all(t)(xor{time(y,t),time(x,t)})})
  wff1!:  all(x)(meeting(x)
                 => (xor{time(x,afternoon),time(x,morning)}))

: time(facultyMeeting,?t)?
  wff11!:  ~time(facultyMeeting,afternoon)
  wff8!:  time(facultyMeeting,morning)
```

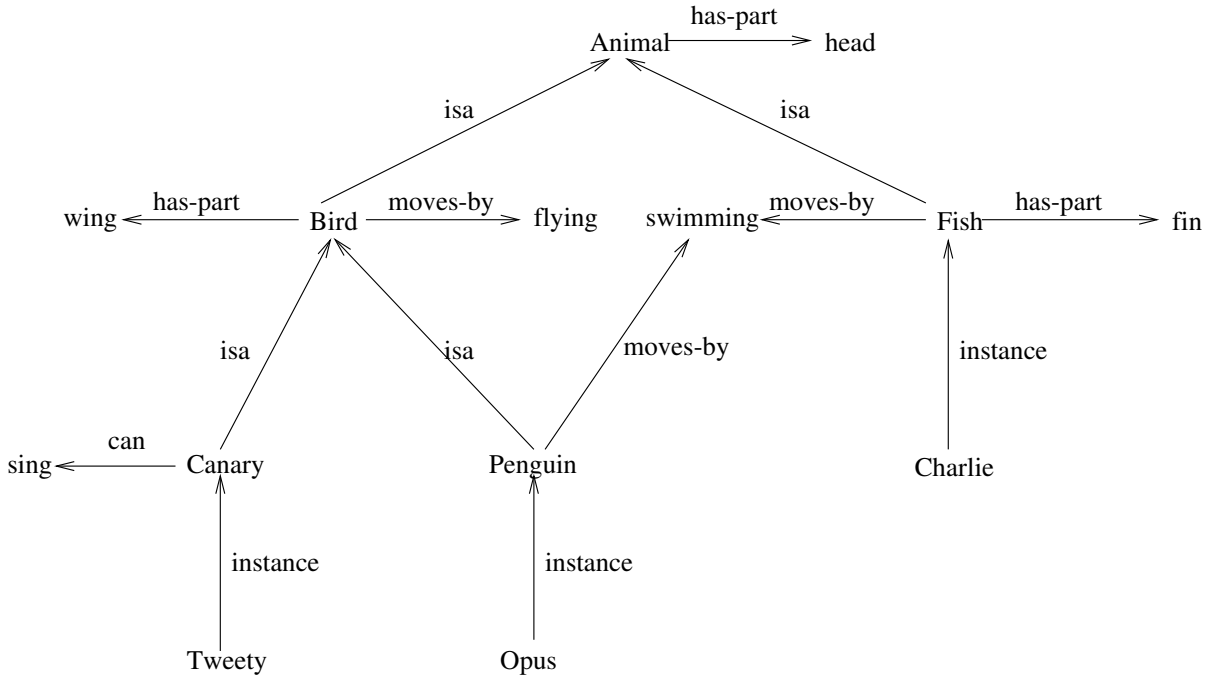
Pat's Schedule

```
: set-default-context patSchedule
((assertions (wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (patSchedule default-defaultct)) (kinconsistent nil))

: time(facultyMeeting,?t)?

  A contradiction was detected within context patSchedule.
  The contradiction involves the newly derived proposition:
      wff8!:  time(facultyMeeting,morning)

  and the previously existing proposition:
      wff9!:  ~time(facultyMeeting,morning)

  You have the following options:
   1. [C]ontinue anyway, knowing that a contradiction is derivable;
   2. [R]e-start the exact same run in a different context which is
      not inconsistent;
   3. [D]rop the run altogether.

  (please type c, r or d)
=><= d
```

Resulting Contexts

```
: describe-context stuSchedule
((assertions (wff6 wff4 wff3 wff2 wff1)) (named (stuSchedule))
 (kinconsistent nil))

: describe-context tonySchedule
((assertions (wff7 wff5 wff3 wff2 wff1)) (named (tonySchedule))
 (kinconsistent nil))

: describe-context patSchedule
((assertions (wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (patSchedule default-defaultct)) (kinconsistent t))
```

8.6 SNePS as a Network: Semantic Networks



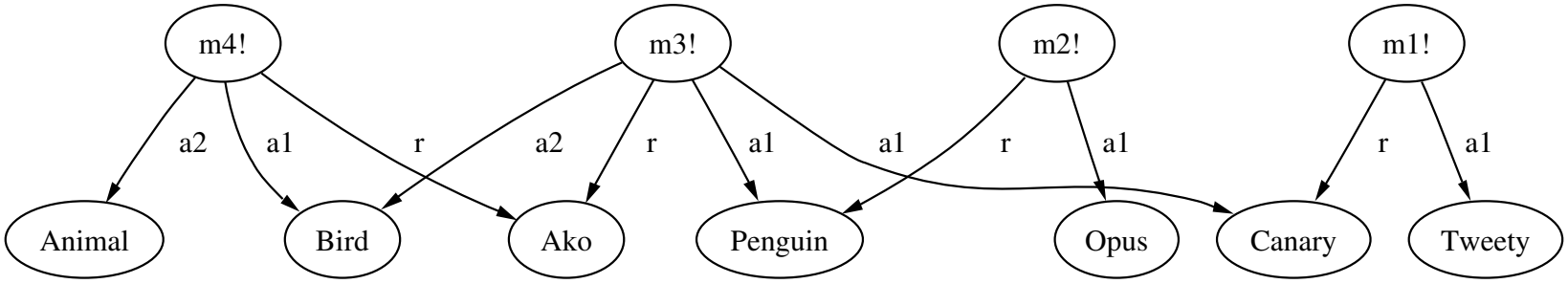
Some psychological evidence.

More efficient search than logical inference.

Unclear semantics.

SNePS as a Network

```
: clearkb
: Canary(Tweety).
: Penguin(Opus).
: Ako({Canary,Penguin}, Bird).
: Ako(Bird, Animal).
: show
```



Defining Case Frames

```
: set-mode-3
Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.
...

: define-frame Canary(class member) "[member] is a [class]"
Canary(x1) will be represented by {<class, Canary>, <member, x1>}

: define-frame Penguin(class member) "[member] is a [class]"
Penguin(x1) will be represented by {<class, Penguin>, <member, x1>}

: define-frame Ako(nil subclass superclass) "Every [subclass] is a [superclas
Ako(x1, x2) will be represented by {<subclass, x1>, <superclass, x2>}
```

Entering the KB

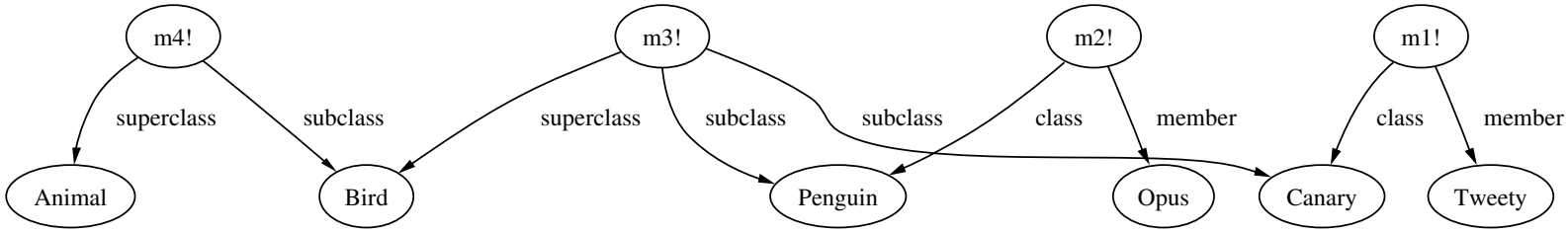
```
: Canary(Tweety).
  wff1!:  Canary(Tweety)

: Penguin(Opus).
  wff2!:  Penguin(Opus)

: Ako({Canary, Penguin}, Bird).
  wff3!:  Ako({Penguin,Canary},Bird)

: Ako(Bird, Animal).
  wff4!:  Ako(Bird,Animal)
```

The Knowledge Base

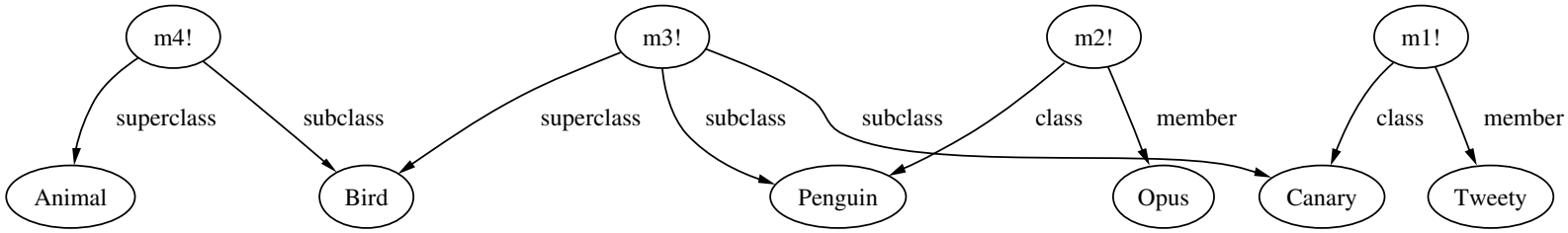
```
: list-terms
  wff1!:  Canary(Tweety)
  wff2!:  Penguin(Opus)
  wff3!:  Ako({Penguin,Canary},Bird)
  wff4!:  Ako(Bird,Animal)

: describe-terms
Tweety is a Canary.
Opus is a Penguin.
Every Penguin and Canary is a Bird.
Every Bird is a Animal.
```

The Network

```
: show
```



Path-Based Inference




```
: define-path class (compose class
                           (kstar (compose subclass- ! superclass)))
class implied by the path (compose class
                                  (kstar
                                   (compose subclass- ! superclass)))
class- implied by the path (compose
                                   (kstar (compose superclass-
                                                   ! subclass))
                                   class-)
```

Using Path-Based Inference

```
: list-asserted-wffs
  wff4!:  Ako(Bird,Animal)
  wff3!:  Ako({Penguin,Canary},Bird)
  wff2!:  Penguin(Opus)
  wff1!:  Canary(Tweety)

: define-frame Animal(class member) "[member] is a [class]"
Animal(x1) will be represented by {<class, Animal>, <member, x1>}

: trace inference
Tracing inference.

: Animal(Tweety)?
I wonder if  wff5:   Animal(Tweety)
holds within the BS defined by context default-defaultct
I know  wff1!:  Canary(Tweety)
  wff5!:  Animal(Tweety)
```

Rules About Functions in Mode 3

```
: set-mode-3
: define-frame WestOf(relation domain range)
: define-frame isAbove(relation domain range)
: define-frame Likes(relation liker likee)

: define-frame r(relation domain range)
: define-frame anti-symmetric(nil antisymm)

: all(r)(anti-symmetric(r) => all(x,y)(r(x,y) => ~r(y,x))).
  wff1!:  all(r)(anti-symmetric(r) => (all(y,x)(r(x,y) => (~r(y,x)))))

: anti-symmetric({WestOf, isAbove, Likes}).

: WestOf(Buffalo,Rochester).
: isAbove(penthouse37,lobby37).
: Likes(Betty,Tom).

: WestOf(?x,?y)?
  wff9!:  ~WestOf(Rochester,Buffalo)
  wff3!:  WestOf(Buffalo,Rochester)

: isAbove(?x,?y)?
  wff13!:  ~isAbove(lobby37,penthouse37)
  wff4!:  isAbove(penthouse37,lobby37)

: Likes(?x,?y)?
  wff5!:  Likes(Betty,Tom)
```

Procedural Attachment in SNePS

```
cl-user(3): (snepslog)
: load /projects/snwiz/Libraries/expressions.snepslog

: define-frame Value(nil obj val) "the value of [obj] is [val]"

: define-frame radius(nil radiusof) "the radius of [radiusof]"

: define-frame volume(nil volumeof) "the volume of [volumeof]"

: all(x,r,p)({Value(radius(x), r), Value(pi,p)}
          &=> all(v)(is(v,/(*(4.0,*(p,*(r,*(r,r)))),3.0))
                     => Value(volume(x),v))).

: Value(pi,3.14159).

: Value(radius(sphere1), 9.0).

: Value(volume(sphere1), ?x)?
  wff13!:  Value(volume(sphere1),3053.6257)
```

8.7 SNeRE: The SNePS Rational Engine

Motivation

Coming to believe something

is different from acting.

Prolog Searches In Order
The KB

```
| ?- [user].
% consulting user...
| q(X) :- q1(X), q2(X).
| q1(X) :- p(X), s(X).
| q2(X) :- r(X), s(X).
| s(X) :- t(X).
| p(a).
| r(a).
| t(a).
|
% consulted user in module user, 0 msec 1592 bytes
yes
```

Prolog Searches In Order
The Run

```
| ?- trace.
% The debugger will first creep -- showing everything (trace)
yes
% trace
| ?- q(a).
        1      1 Call: q(a) ?
        2      2 Call: q1(a) ?
        3      3 Call: p(a) ?
        3      3 Exit: p(a) ?
        4      3 Call: s(a) ?
        5      4 Call: t(a) ?
        5      4 Exit: t(a) ?
        4      3 Exit: s(a) ?
        2      2 Exit: q1(a) ?
        6      2 Call: q2(a) ?
        7      3 Call: r(a) ?
        7      3 Exit: r(a) ?
        8      3 Call: s(a) ?
        9      4 Call: t(a) ?
        9      4 Exit: t(a) ?
        8      3 Exit: s(a) ?
        6      2 Exit: q2(a) ?
        1      1 Exit: q(a) ?
yes
```

SNePS Avoids Extra Search

The KB

```
: clearkb
Knowledge Base Cleared

: all(x)({q1(x), q2(x)} &=> q(x)).
: all(x)({p(x), s(x)} &=> q1(x)).
: all(x)({r(x), s(x)} &=> q2(x)).
: all(x)(t(x) => s(x)).

: p(a).
: r(a).
: t(a).
```

SNePS Avoids Extra Search
The Search

```
: trace inference
Tracing inference.

: q(a)?
I wonder if  wff8:   q(a)
I wonder if  wff10:   q2(a)
I wonder if  wff12:   q1(a)
I wonder if  wff14:   s(a)
I wonder if  wff6!:  r(a)
I wonder if  wff14:   s(a)
I wonder if  wff5!:  p(a)
I know  wff6!:  r(a)
I know  wff5!:  p(a)
I wonder if  wff7!:  t(a)
I know  wff7!:  t(a)
```

SNePS Avoids Extra Search
The Answers

```
Since  wff4!:  all(x)(t(x) => s(x))
and  wff7!:  t(a)
I infer  wff14:   s(a)

Since  wff3!:  all(x)({s(x),r(x)} &=> {q2(x)})
and  wff14!:  s(a)
and  wff6!:  r(a)
I infer  wff10:   q2(a)

Since  wff2!:  all(x)({s(x),p(x)} &=> {q1(x)})
and  wff14!:  s(a)
and  wff5!:  p(a)
I infer  wff12:   q1(a)

Since  wff1!:  all(x)({q2(x),q1(x)} &=> {q(x)})
and  wff10!:  q2(a)
and  wff12!:  q1(a)
I infer  wff8:   q(a)

  wff8!:  q(a)
```

Primitive Acts

```
: set-mode-3
Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.
...

: define-frame say(action line)
say(x1) will be represented by {<action, say>, <line, x1>}

: ^^
--> (define-primaction sayaction ((line))
      (format sneps:outunit "~A" line))
sayaction

--> (attach-primaction say sayaction)
t

--> ^^

: perform say("Hello world")
Hello world
```

Effects: The KB

```
: set-mode-3
Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.
...
Effect(x1, x2) will be represented by {<act, x1>, <effect, x2>}
...

: define-frame say (action line)
: define-frame said (act agent object)
: define-frame Utterance (class member)
: ^^
--> (define-primaction sayaction ((line))
      (format sneps:outunit "~A" line))
sayaction
-->
(attach-primaction say sayaction)
t
--> ^^
: Utterance("Hello world").
: all(x)(Utterance(x) => Effect(say(x), said(I,x))).
```

Effects: The Run

```
: list-asserted-wffs
  wff2!:  all(x)(Utterance(x) => Effect(say(x),said(I,x)))
  wff1!:  Utterance(Hello world)

: perform say("Hello world")
Hello world

: list-asserted-wffs
  wff5!:  Effect(say(Hello world),said(I,Hello world))
  wff4!:  said(I,Hello world)
  wff2!:  all(x)(Utterance(x) => Effect(say(x),said(I,x)))
  wff1!:  Utterance(Hello world)
```

Defined Acts

```
: set-mode-3
...
ActPlan(x1, x2) will be represented by {<act, x1>, <plan, x2>}
...

: define-frame say (action part1 part2)
: define-frame greet (action object)
: define-frame Person (class member)

: ^^
--> (define-primaction sayaction ((part1) (part2))
      (format sneps:outunit "~A ~A~%"
              part1 part2))
sayaction
-->
(attach-primaction say sayaction)
t
--> ^^

: all(x)(Person(x) => ActPlan(greet(x), say(Hello,x))).
: Person(Mike).

: perform greet(Mike).
Hello Mike
```

Other Propositions about Acts

GoalPlan(*p*, *a*)

Precondition(*a*, *p*)

Control Acts

achieve(*p*)

do-all({*a1, ..., an*})

do-one({*a1, ..., an*})

snif({if(*p1,a1*), ..., if(*pn,an*)[, else(*da*)]})

sniterate({if(*p1,a1*), ..., if(*pn,an*)[, else(*da*)]})

snsequence(*a1, a2*)

withall(*x, p(x), a(x)*[, *da*])

withsome(*x, p(x), a(x)*[, *da*])

Must use attach-primaction on whichever you want to use.

Policies

`ifdo(`*p*`, `*a*`)`

`whendo(`*p*`, `*a*`)`

`wheneverdo(`*p*`, `*a*`)`

Mental Acts

believe(p)

disbelieve(p)

adopt(p)

unadopt(p)

The Execution Cycle

```
perform(act):
   pre := {p | Precondition(act,p)};
   notyet := pre - {p | p ∈ pre & ⊢ p};
   if notyet ≠ nil
      then perform(snsequence(do-all({a | p ∈ notyet
                                          & a = achieve(p)}),
                              act))
      else {effects := {p | Effect(act,p)};
            if act is primitive
               then apply(primitive-function(act), objects(act));
               else perform(do-one({p | ActPlan(act,p)}))
            believe(effects)}
```

Examples

SNePSLOG demo #7

/projects/robot/Karel/ElevatorWorld/elevator.snepslog

/projects/robot/Karel/DeliveryWorld/DeliveryAgent.snepslog

/projects/robot/Karel/WumpusWorld/WWAgent.snepslog

/projects/robot/Fevahr/Ascii/afevahr.snepslog

/projects/robot/Fevahr/Java/jfevahr.snepslog

/projects/robot/Greenfoot/ElevatorWorld/sneps/elevator.snepslog

/projects/robot/Greenfoot/WumpusWorld/sneps/WWAgent.snepslog