An Introduction to SNePS

Stuart C. Shapiro
Department of Computer Science and Engineering
and Center for Cognitive Science
201 Bell Hall
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
shapiro@cse.buffalo.edu

March 24, 2009

Contents

1 Overview

- A knowledge representation, reasoning, and acting system.
- Over 30 years of development.
- 65 people involved in its development.
- Academic, not "productized", system.
- Constantly being improved: suggestions and help appreciated.
- Implementation language: ANSI Common Lisp.
- Logic-based, frame-based, and network-based.
- Interfaces: **SNePSLOG**; SNePSUL; Fragments of English.

The SNePS sessions shown in this document have been extracted directly from actual runs. To facilitate editing this document for new versions of SNePS, these sessions have been edited only to break exceptionally long lines. Eventually, this last sentence will be replaced by the sentences:

> They have been edited to improve line breaks, and to eliminate some blank lines. The response to each input ends with a report of the CPU time used by processing the input. These lines have also been deleted. Some comments have also been deleted, especially when they are replaced by text in this document.

This document is not yet finished. In several cases, a section only contains the name of a demo file. That file is available, and may be run to illustrate the material that will be discussed in the section when this document is completed.

2 Atomic Formulas in SNePSLOG

The SNePSLOG parser does not recognize a single proposition symbol as a formula (also called a "wff," "proposition," or "belief")[1]. So a SNePSLOG atomic formula consists of a predicate symbol followed by zero or more arguments, surrounded by parentheses, and separated by commas.

2.1 Simple Assertions

The SNePSLOG prompt is "`:`". User input can be a command, assertion, or query. Comments can also be entered, preceded by one or more semicolons. To load a file, use the `demo` command, and give it the name of the file to be loaded. The full name of each file used in this document is shown so that the reader may try it for him/herself.

```
: demo /projects/shapiro/Sneps/SnepsIntro/propAssertions.snepslog

File /projects/shapiro/Sneps/SnepsIntro/propAssertions.snepslog is now the source of input


 CPU time : 0.00

:
;;; Empty and initialize the SNePS knowledge base.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;;To enter an assertion (belief) into the knowledge base, type it in
;;;followed by a period.  The smallest assertion is a predicate symbol
;;;followed by an empty pair of parentheses.  Mixed case is recognized
;;;when the ACL version of Common Lisp is being used.  The assertion is
;;;echoed by the system preceded by its "wffName" and an exclamation mark
;;;to show that it is asserted.
;;;
;;;CarPool World will be used for examples of Propositional Logic.
BettyDrivesTom().

  wff1!:  BettyDrivesTom()

 CPU time : 0.00

: BettyIsDriver().

  wff2!:  BettyIsDriver()

 CPU time : 0.00

: TomIsPassenger().

  wff3!:  TomIsPassenger()
```

[1] This is actually a restriction of SNePS, not just of SNePSLOG.

```
 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/propAssertions.snepslog demonstration.
```

Predicate Logic will be illustrated by a Great Lakes domain.

```
: demo /projects/shapiro/Sneps/SnepsIntro/assertions.snepslog

File /projects/shapiro/Sneps/SnepsIntro/assertions.snepslog is now the source of input.


 CPU time : 0.00

: ;;; Empty the Knowledge Base
clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; There are five lakes: Superior, Michigan, Huron, Erie, and Ontario.
Lake(LakeSuperior).

  wff1!:  Lake(LakeSuperior)

 CPU time : 0.00

: Lake(LakeMichigan).

  wff2!:  Lake(LakeMichigan)

 CPU time : 0.00

: Lake(LakeHuron).

  wff3!:  Lake(LakeHuron)

 CPU time : 0.00

: Lake(LakeErie).

  wff4!:  Lake(LakeErie)

 CPU time : 0.00

: Lake(LakeOntario).

  wff5!:  Lake(LakeOntario)
```

```
 CPU time : 0.00

: ;;; The St. Lawrence is a river.
River(StLawrenceRiver).

  wff6!:  River(StLawrenceRiver)

 CPU time : 0.00

: ;;; Multiple arguments are separated by commas.
;;;
;;; Lakes Superior and Michigan feed Lake Huron.
Feeds(LakeSuperior,LakeHuron).

  wff7!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

: Feeds(LakeMichigan, LakeHuron).

  wff8!:  Feeds(LakeMichigan,LakeHuron)

 CPU time : 0.00

: ;;; Lake Huron feeds Lake Erie.
Feeds(LakeHuron, LakeErie).

  wff9!:  Feeds(LakeHuron,LakeErie)

 CPU time : 0.00

: ;;; Lake Ontario feeds the St. Lawrence River.
Feeds(LakeOntario, StLawrenceRiver).

  wff10!:  Feeds(LakeOntario,StLawrenceRiver)

 CPU time : 0.00

:  ;;; Lake Huron doesn't feed the St. Lawrence river.
DoesntFeed(LakeHuron, StLawrenceRiver).

  wff11!:  DoesntFeed(LakeHuron,StLawrenceRiver)

 CPU time : 0.00

:
;;; The lakes border on various states.
Borders(LakeSuperior, Minnesota).

  wff12!:  Borders(LakeSuperior,Minnesota)
```

```
 CPU time : 0.00

: Borders(LakeMichigan, Illinois).

  wff13!:  Borders(LakeMichigan,Illinois)

 CPU time : 0.00

: Borders(LakeHuron, Michigan).

  wff14!:  Borders(LakeHuron,Michigan)

 CPU time : 0.00

: Borders(LakeErie, Michigan).

  wff15!:  Borders(LakeErie,Michigan)

 CPU time : 0.00

:
;;; Lakes are at various latitudes and longitudes.
;;;For purposes of this introductory document, each lake has been
;;;assigned a single longitude and latitude.  These locations are
;;;actually somewhere in, and approximately at the center of, each lake.
;;;
;;;A predicate symbol or individual constant may be any Lisp symbol.
Latitude(LakeSuperior, 47N).

  wff16!:  Latitude(LakeSuperior,47N)

 CPU time : 0.00

: Latitude(LakeMichigan, 44N).

  wff17!:  Latitude(LakeMichigan,44N)

 CPU time : 0.00

: Latitude(LakeHuron, 45N).

  wff18!:  Latitude(LakeHuron,45N)

 CPU time : 0.00

: Longitude(LakeSuperior, 88W).

  wff19!:  Longitude(LakeSuperior,88W)

 CPU time : 0.01
```

```
: Longitude(LakeMichigan, 87W).

  wff20!:  Longitude(LakeMichigan,87W)

 CPU time : 0.00

: Longitude(LakeHuron, 82W).

  wff21!:  Longitude(LakeHuron,82W)

 CPU time : 0.00

:
;;;A predicate symbol or individual constant may also be any integer.
;;;However, these will be treated by SNePS as symbols.  SNePS does not
;;;know anything about numbers.
;;; Lakes have various areas (in square miles).
Area(LakeSuperior, 31968).

  wff22!:  Area(LakeSuperior,31968)

 CPU time : 0.00

: Area(LakeMichigan, 22316).

  wff23!:  Area(LakeMichigan,22316)

 CPU time : 0.00

: Area(LakeHuron, 23011).

  wff24!:  Area(LakeHuron,23011)

 CPU time : 0.00

: Area(LakeErie, 9922).

  wff25!:  Area(LakeErie,9922)

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/assertions.snepslog demonstration.
```

2.2 Simple Queries: `?` and `askwh`

There are two ways to ask SNePS simple queries: a formula followed by a question mark; and via the `askwh` command. These are demonstrated, and the difference between them is explained in this section.

```
: demo /projects/shapiro/Sneps/SnepsIntro/queries.snepslog
```

```
File /projects/shapiro/Sneps/SnepsIntro/queries.snepslog is now the source of input.


 CPU time : 0.00

: ;;;The load command is for loading a file of assertions.  It does not
;;;show the contents of the file being loaded.
;;; Load the Simple Assertions
load /projects/shapiro/Sneps/SnepsIntro/assertions.snepslog


 CPU time : 0.05

: ;;; We can see that the file was loaded by using the list-wffs command,
;;; which lists all the formulas that have been asserted.
list-wffs
  wff25!:  Area(LakeErie,9922)
  wff24!:  Area(LakeHuron,23011)
  wff23!:  Area(LakeMichigan,22316)
  wff22!:  Area(LakeSuperior,31968)
  wff21!:  Longitude(LakeHuron,82W)
  wff20!:  Longitude(LakeMichigan,87W)
  wff19!:  Longitude(LakeSuperior,88W)
  wff18!:  Latitude(LakeHuron,45N)
  wff17!:  Latitude(LakeMichigan,44N)
  wff16!:  Latitude(LakeSuperior,47N)
  wff15!:  Borders(LakeErie,Michigan)
  wff14!:  Borders(LakeHuron,Michigan)
  wff13!:  Borders(LakeMichigan,Illinois)
  wff12!:  Borders(LakeSuperior,Minnesota)
  wff11!:  DoesntFeed(LakeHuron,StLawrenceRiver)
  wff10!:  Feeds(LakeOntario,StLawrenceRiver)
  wff9!:  Feeds(LakeHuron,LakeErie)
  wff8!:  Feeds(LakeMichigan,LakeHuron)
  wff7!:  Feeds(LakeSuperior,LakeHuron)
  wff6!:  River(StLawrenceRiver)
  wff5!:  Lake(LakeOntario)
  wff4!:  Lake(LakeErie)
  wff3!:  Lake(LakeHuron)
  wff2!:  Lake(LakeMichigan)
  wff1!:  Lake(LakeSuperior)


 CPU time : 0.01

:
;;; T/F questions
;;; =============
;;;The user may ask SNePS a true/false question by entering a SNePSLOG
;;;formula followed by a question mark.  If the formula is asserted in
```

```
;;;the knowledge base, the system prints the assertion preceded by its
;;;wffName and an exclamation mark.
;;; True questions
;;; --------------
;;; Is Superior a lake?
Lake(LakeSuperior)?

  wff1!:  Lake(LakeSuperior)

 CPU time : 0.00

: ;;; Does Lake Michigan border Illinois?
Borders(LakeMichigan, Illinois)?

  wff13!:  Borders(LakeMichigan,Illinois)

 CPU time : 0.00

: ;;; False questions
;;; ---------------
;;;If the formula of the query is not asserted in the knowledge base,
;;;SNePS just goes on to the next prompt.  SNePS uses the Open World
;;;Assumption, that just because a proposition is not asserted in the
;;;knowledge base that doesn't make it false.  The assumption that
;;;everything that is not stored as true is false is called the Closed
;;;World Assumption.
;;; Does Lake Huron border on Minnesota?
Borders(LakeHuron, Minnesota)?


 CPU time : 0.00

: ;;; Does Lake Ontario feed the St Lawrence river?
Feeds(LakeOntario, StLawrenceRiver)?

  wff10!:  Feeds(LakeOntario,StLawrenceRiver)

 CPU time : 0.00

: ;;; Does Lake Huron feed the St Lawrence river?
Feeds(LakeHuron, StLawrenceRiver)?


 CPU time : 0.00

: ;;; Wh questions
;;; ============
;;;Wh questions may be asked by entering a wff with one or more arguments
;;;replaced by a query variable, which is any Lisp symbol that begins
;;;with a question mark, and following the wff with a question mark as
;;;for true/false questions.  If an answer is found, the belief will be
```

```
;;;printed with the query variable replaced by the correct constant.
;;; What does Lake Superior border?
Borders(LakeSuperior, ?s)?

  wff12!:  Borders(LakeSuperior,Minnesota)

 CPU time : 0.00

: ;;; What borders on Michigan?
Borders(?L, Michigan)?

  wff15!:  Borders(LakeErie,Michigan)
  wff14!:  Borders(LakeHuron,Michigan)

 CPU time : 0.00

: ;;; What lakes are "31,968 sq. mi." in area?
Area(?L, 31968)?

  wff22!:  Area(LakeSuperior,31968)

 CPU time : 0.01

: ;;;In SNePSLOG, a predicate symbol may also be replaced by a query
;;;variable to ask about the relationship between or among various entities.
;;; Does Lake Huron feed the St. Lawrence river or not?
?r(LakeHuron, StLawrenceRiver)?

  wff11!:  DoesntFeed(LakeHuron,StLawrenceRiver)

 CPU time : 0.00

: ;;;A question formed by a formula followed by a question mark is
;;;answered, if at all, by a list of formulas.  Think of that as
;;;answering a question with a complete sentence.  If all that is wanted
;;;is a list of the satisfying constants, use the askwh command.
;;; List the lakes.
askwh Lake(?x)

((?x . LakeOntario))
((?x . LakeErie))
((?x . LakeHuron))
((?x . LakeMichigan))
((?x . LakeSuperior))

 CPU time : 0.00

: ;;; List the lakes that feed the St. Lawrence river.
askwh Feeds(?x, StLawrenceRiver)

((?x . LakeOntario))
```

```
 CPU time : 0.00

: ;;; List the lakes at latitude 47N.
askwh Latitude(?L, 47N)

((?L . LakeSuperior))

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/queries.snepslog demonstration.
```

2.3 Functional Terms and Conjunctive Queries

So far, arguments to SNePSLOG predicates have been individual constants or query variables. In this section we will see that terms may also be functional terms. We will also see how to ask conjunctive queries.

```
: demo /projects/shapiro/Sneps/SnepsIntro/functions.snepslog

File /projects/shapiro/Sneps/SnepsIntro/functions.snepslog is now the source of input.


 CPU time : 0.01

: ;;; Empty the Knowledge Base
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;;We will recreate a slightly different version of the same knowledge base
;;;used above.
;;; There are five lakes: Superior, Michigan, Huron, Erie, and Ontario.
Lake(LakeSuperior).

  wff1!:  Lake(LakeSuperior)

 CPU time : 0.00

: Lake(LakeMichigan).

  wff2!:  Lake(LakeMichigan)

 CPU time : 0.00

: Lake(LakeHuron).

  wff3!:  Lake(LakeHuron)
```

```
 CPU time : 0.00

: Lake(LakeErie).

  wff4!:  Lake(LakeErie)

 CPU time : 0.00

: Lake(LakeOntario).

  wff5!:  Lake(LakeOntario)

 CPU time : 0.00

: ;;; The St. Lawrence is a river.
River(StLawrenceRiver).

  wff6!:  River(StLawrenceRiver)

 CPU time : 0.00

:
;;; Lakes Superior and Michigan feed Lake Huron.
Feeds(LakeSuperior,LakeHuron).

  wff7!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

: Feeds(LakeMichigan, LakeHuron).

  wff8!:  Feeds(LakeMichigan,LakeHuron)

 CPU time : 0.00

: ;;; Lake Huron feeds Lake Erie.
Feeds(LakeHuron, LakeErie).

  wff9!:  Feeds(LakeHuron,LakeErie)

 CPU time : 0.00

: ;;; Lake Ontario feeds the St. Lawrence River
Feeds(LakeOntario, StLawrenceRiver).

  wff10!:  Feeds(LakeOntario,StLawrenceRiver)

 CPU time : 0.00

:  ;;; Lake Huron doesn’t feed the St. Lawrence river.
DoesntFeed(LakeHuron, StLawrenceRiver).
```

```
  wff11!:  DoesntFeed(LakeHuron,StLawrenceRiver)

 CPU time : 0.01

:
;;; The lakes border on various states.
Borders(LakeSuperior, Minnesota).

  wff12!:  Borders(LakeSuperior,Minnesota)

 CPU time : 0.00

: Borders(LakeMichigan, Illinois).

  wff13!:  Borders(LakeMichigan,Illinois)

 CPU time : 0.00

: Borders(LakeHuron, Michigan).

  wff14!:  Borders(LakeHuron,Michigan)

 CPU time : 0.00

: Borders(LakeErie, Michigan).

  wff15!:  Borders(LakeErie,Michigan)

 CPU time : 0.00

:
;;;Now, instead of giving the latitude and longitude of each lake as
;;;separate assertions, we will use the functional term, latLong(x,y), to
;;;represent the location at latitude x and longitude y.
;;; Lakes are at various latitudes and longitudes.
Location(LakeSuperior,  latLong(47N, 88W)).

  wff17!:  Location(LakeSuperior,latLong(47N,88W))

 CPU time : 0.00

: Location(LakeMichigan,  latLong(44N, 87W)).

  wff19!:  Location(LakeMichigan,latLong(44N,87W))

 CPU time : 0.00

: Location(LakeHuron,     latLong(45N, 82W)).

  wff21!:  Location(LakeHuron,latLong(45N,82W))
```

```
 CPU time : 0.00

: Location(LakeErie,      latLong(42N, 81W)).

  wff23!:  Location(LakeErie,latLong(42N,81W))

 CPU time : 0.00

: Location(LakeOntario,   latLong(44N, 78W)).

  wff25!:  Location(LakeOntario,latLong(44N,78W))

 CPU time : 0.00

:
;;;We will give the areas as before.
;;; Lakes have various areas.
Area(LakeSuperior,  31968).

  wff26!:  Area(LakeSuperior,31968)

 CPU time : 0.01

: Area(LakeHuron,     23011).

  wff27!:  Area(LakeHuron,23011)

 CPU time : 0.01

: Area(LakeMichigan,  22316).

  wff28!:  Area(LakeMichigan,22316)

 CPU time : 0.00

: Area(LakeErie,      9922).

  wff29!:  Area(LakeErie,9922)

 CPU time : 0.00

: Area(LakeOntario,   7320).

  wff30!:  Area(LakeOntario,7320)

 CPU time : 0.00

:
;;; Questions
;;; =========
```

```
;;;Functional terms can be answers to questions, and may be used in
;;;questions.  When used in a question, a functional term may have some
;;;constant arguments and some query variable arguments.
;;;Where is lake Superior?
Location(LakeSuperior, ?x)?

  wff17!:  Location(LakeSuperior,latLong(47N,88W))

 CPU time : 0.00

: ;;; What lakes are at latitude 44N?
Location(?L, latLong(44N, ?long))?

  wff25!:  Location(LakeOntario,latLong(44N,78W))
  wff19!:  Location(LakeMichigan,latLong(44N,87W))

 CPU time : 0.00

: ;;; List the lakes at latitude 44N.
askwh Location(?L, latLong(44N, ?long))

((?long . 78W)(?L . LakeOntario))
((?long . 87W)(?L . LakeMichigan))

 CPU time : 0.00

:
;;;A conjunctive query can be formed from several regular queries
;;;connected by and.  Notice that, while two lakes are at latitude 44N,
;;;only one of those borders on Illinois.
;;; What borders on Illinois and is at latitude 44N?
Borders(?L, Illinois) and Location(?L, latLong(44N, ?long))?

  wff31!:  Location(LakeMichigan,latLong(44N,87W)) and Borders(LakeMichigan,Illinois)

 CPU time : 0.05

:

End of /projects/shapiro/Sneps/SnepsIntro/functions.snepslog demonstration.
```

2.4 Reified Propositions

We have been saying that SNePSLOG symbols such as `Borders`, used in expressions such as `Borders(LakeSuperior, Minnesota)`, are predicate symbols. In classical first-order logic, expressions formed by predicate symbols and their arguments are sentences, and may not be used as arguments in other sentences. However, in SNePSLOG, symbols such as `Borders` are actually function symbols just like `latLong`, and the expressions formed by them and their arguments are functional terms just like `latLong(47N, 88W)`, and may be used as arguments to other function symbols without leaving first-order logic. While the functional term `latLong(47N, 88W)` denotes a geographical location, the functional term `Borders(LakeSuperior, Minnesota)` denotes a proposition. We will demonstrate this facility in this section.

```
: demo /projects/shapiro/Sneps/SnepsIntro/reified.snepslog

File /projects/shapiro/Sneps/SnepsIntro/reified.snepslog is now the source of input.


 CPU time : 0.01

: ;;; Load the same file used in the previous section.
load /projects/shapiro/Sneps/SnepsIntro/functions.snepslog


 CPU time : 0.06

:
;;; Attribute two Borders beliefs to Betty.
;;; Betty believes that Lake Superior borders on Minnesota.
Believes(Betty, Borders(LakeSuperior, Minnesota)).

  wff32!:  Believes(Betty,Borders(LakeSuperior,Minnesota))

 CPU time : 0.00

: ;;; Betty believes that Lake Michigan borders on Pennsylvania.
Believes(Betty, Borders(LakeMichigan, Pennsylvania)).

  wff34!:  Believes(Betty,Borders(LakeMichigan,Pennsylvania))

 CPU time : 0.00

: ;;; What does Betty believe?
Believes(Betty, ?p)?

  wff34!:  Believes(Betty,Borders(LakeMichigan,Pennsylvania))
  wff32!:  Believes(Betty,Borders(LakeSuperior,Minnesota))

 CPU time : 0.00

: ;;; What does Betty believe that is true?
Believes(Betty, ?p) and ?p?

  wff35!:  Believes(Betty,Borders(LakeSuperior,Minnesota))
           and Borders(LakeSuperior,Minnesota)

 CPU time : 0.02

:

End of /projects/shapiro/Sneps/SnepsIntro/reified.snepslog demonstration.
```

2.5 Unique Names Assumption and Uniqueness Principle

Every expression stored in the knowledge base is represented in one place. If you enter an expression that looks exactly like one you entered before, the previous one is used. This can be seen by comparing wffNames, as shown here.

```
: demo /projects/shapiro/Sneps/SnepsIntro/unique.snepslog

File /projects/shapiro/Sneps/SnepsIntro/unique.snepslog is now the source of input.


 CPU time : 0.00

: ;;; Empty the Knowledge Base
clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; Make some assertions.
Lake(LakeErie).

  wff1!:  Lake(LakeErie)

 CPU time : 0.00

: Lake(LakeOntario).

  wff2!:  Lake(LakeOntario)

 CPU time : 0.00

:
;;; Make one again.  The previous one is used.  (Compare wffNames.)
Lake(LakeErie).

  wff1!:  Lake(LakeErie)

 CPU time : 0.00

:
;;;The wff numbers are assigned in order.  If an expression contains a
;;;subexpressions, the wff numbers are assigned bottom up.  This
;;;assertion creates two new expressions:
Believes(Betty, Lake(LakeHuron)).

  wff4!:  Believes(Betty,Lake(LakeHuron))

 CPU time : 0.00

:
```

```
;;;However, this assertion uses an old proposition, so only one new
;;;expression is created:
Believes(Betty, Lake(LakeErie)).

  wff5!:  Believes(Betty,Lake(LakeErie))

 CPU time : 0.00

:
;;; We can use a wffName instead of spelling out the expression:
Believes(Betty, wff2).

  wff6!:  Believes(Betty,Lake(LakeOntario))

 CPU time : 0.00

:
;;; List all asserted wffs, and recheck the wffNames.
list-wffs
  wff6!:  Believes(Betty,Lake(LakeOntario))
  wff5!:  Believes(Betty,Lake(LakeErie))
  wff4!:  Believes(Betty,Lake(LakeHuron))
  wff2!:  Lake(LakeOntario)
  wff1!:  Lake(LakeErie)


 CPU time : 0.00

: ;;; To see the wffNames of all the terms, including those not asserted use list-terms.
;;; The wffNames of terms that are not asserted are not followed by an exclamation mark.
list-terms

  wff1!:  Lake(LakeErie)
  wff2!:  Lake(LakeOntario)
  wff3:   Lake(LakeHuron)
  wff4!:  Believes(Betty,Lake(LakeHuron))
  wff5!:  Believes(Betty,Lake(LakeErie))
  wff6!:  Believes(Betty,Lake(LakeOntario))

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/unique.snepslog demonstration.
```

3 Reduction Inference

3.1 Reduction, One Kind of Inference

There are three kinds of inference supported by SNePS: reduction inference, formula-based inference, and path-based inference. We now have presented enough background to discuss reduction inference.

We previously saw the function symbol `Feeds` used to form expressions like `Feeds(LakeSuperior,LakeHuron)`, which denotes the proposition that Lake Superior feeds Lake Huron. Wherever an argument is allowed in SNePSLOG, we can put either a term or a set of terms, which is entered as a sequence of terms, separated by commas, and surrounded by curly braces. For example,

```
: Feeds({LakeSuperior, LakeMichigan}, LakeHuron).
```

denotes the proposition that Lake Superior and Lake Michigan feed Lake Huron.

There are a pair of rules of inference for reduction inference. One of them may be expressed as,

Reduction Inference (a): $\mathtt{P}(\mathtt{a}_1, \ldots, \{\ldots, \mathtt{a}_{ij}, \ldots\}, \ldots, \mathtt{a}_n) \vdash \mathtt{P}(\mathtt{a}_1, \ldots, \mathtt{a}_{ij}, \ldots, \mathtt{a}_n)$

This means that if the proposition $\mathtt{P}(\mathtt{a}_1, \ldots, \{\ldots, \mathtt{a}_{ij}, \ldots\}, \ldots, \mathtt{a}_n)$ is asserted in the knowledge base, then the proposition $\mathtt{P}(\mathtt{a}_1, \ldots, \mathtt{a}_{ij}, \ldots, \mathtt{a}_n)$, may be asserted in the knowledge base also.

Since each argument may be a set, one proposition may express an exponential number of atomic propositions. For example,

```
Isa({LakeHuron, LakeErie, LakeOntario}, {BodyOfWater, Lake}).
```

expresses the proposition that Lakes Huron, Erie, and Ontario are all both bodies of water and lakes.

These features are demonstrated in the following.

```
: demo /projects/shapiro/Sneps/SnepsIntro/reduction.snepslog

File /projects/shapiro/Sneps/SnepsIntro/reduction.snepslog is now the source of input.


 CPU time : 0.00

: ;;; Empty the Knowledge Base
clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; Lakes Superior and Michigan feed Lake Huron.
Feeds({LakeSuperior, LakeMichigan}, LakeHuron).

  wff1!:  Feeds({LakeMichigan,LakeSuperior},LakeHuron)

 CPU time : 0.00

: ;;; Does Lake Superior feed Lake Huron?
Feeds(LakeSuperior, LakeHuron)?

  wff2!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.01

: ;;; What feeds Lake Huron?
Feeds(?L, LakeHuron)?
```

```
  wff3!:  Feeds(LakeMichigan,LakeHuron)
  wff2!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

: ;;; Lakes Huron, Erie, and Ontario are bodies of water and lakes.
Isa({LakeHuron, LakeErie, LakeOntario}, {BodyOfWater, Lake}).

  wff4!:  Isa({LakeOntario,LakeErie,LakeHuron},{Lake,BodyOfWater})

 CPU time : 0.00

: ;;; The St. Lawrence river is a body of water and a river.
Isa(StLawrenceRiver, {BodyOfWater, River}).

  wff5!:  Isa(StLawrenceRiver,{River,BodyOfWater})

 CPU time : 0.00

: ;;; Is Lake Erie a lake?
Isa(LakeErie, Lake)?

  wff6!:  Isa(LakeErie,Lake)

 CPU time : 0.00

: ;;; What kind of body of water is  Lake Huron?
Isa(LakeHuron, {BodyOfWater, ?x})?

  wff7!:  Isa(LakeHuron,{Lake,BodyOfWater})

 CPU time : 0.00

: ;;; List the bodies of water.
askwh Isa(?x, BodyOfWater)

((?x . StLawrenceRiver))
((?x . LakeHuron))
((?x . LakeErie))
((?x . LakeOntario))

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/reduction.snepslog demonstration.
```

3.2 Tracing Inference

One can follow SNePS's reasoning process by turning on inference tracing.

```
: demo /projects/shapiro/Sneps/SnepsIntro/reductionTrace.snepslog
```

```
File /projects/shapiro/Sneps/SnepsIntro/reductionTrace.snepslog is now the source of input


 CPU time : 0.00

: ;;; Empty the Knowledge Base
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; Turn on inference tracing
trace inference
Tracing inference.


 CPU time : 0.00

: ;;; Lakes Superior and Michigan feed Lake Huron.
Feeds({LakeSuperior, LakeMichigan}, LakeHuron).

  wff1!:  Feeds({LakeMichigan,LakeSuperior},LakeHuron)

 CPU time : 0.00

: ;;; Does Lake Superior feed Lake Huron?
Feeds(LakeSuperior, LakeHuron)?

I wonder if  wff2:   Feeds(LakeSuperior,LakeHuron)
holds within the BS defined by context default-defaultct

I know  wff1!:  Feeds({LakeMichigan,LakeSuperior},LakeHuron)

  wff2!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

: ;;; What feeds Lake Huron?
Feeds(?L, LakeHuron)?

I wonder if  p1:   Feeds(L,LakeHuron)
holds within the BS defined by context default-defaultct

I know  wff1!:  Feeds({LakeMichigan,LakeSuperior},LakeHuron)

I know  wff2!:  Feeds(LakeSuperior,LakeHuron)

  wff3!:  Feeds(LakeMichigan,LakeHuron)
  wff2!:  Feeds(LakeSuperior,LakeHuron)
```

```
 CPU time : 0.01

: ;;; Lakes Huron, Erie, and Ontario are bodies of water and lakes.
Isa({LakeHuron, LakeErie, LakeOntario}, {BodyOfWater, Lake}).

  wff4!:  Isa({LakeOntario,LakeErie,LakeHuron},{Lake,BodyOfWater})

 CPU time : 0.00

: ;;; The St. Lawrence river is a body of water and a river.
Isa(StLawrenceRiver, {BodyOfWater, River}).

  wff5!:  Isa(StLawrenceRiver,{River,BodyOfWater})

 CPU time : 0.00

: ;;; Is Lake Erie a lake?
Isa(LakeErie, Lake)?

I wonder if  wff6:   Isa(LakeErie,Lake)
holds within the BS defined by context default-defaultct

I know  wff4!:  Isa({LakeOntario,LakeErie,LakeHuron},{Lake,BodyOfWater})

  wff6!:  Isa(LakeErie,Lake)

 CPU time : 0.00

: ;;; What kind of body of water is  Lake Huron?
Isa(LakeHuron, {BodyOfWater, ?x})?

I wonder if  p2:   Isa(LakeHuron,{x,BodyOfWater})
holds within the BS defined by context default-defaultct

I know  wff4!:  Isa({LakeOntario,LakeErie,LakeHuron},{Lake,BodyOfWater})

  wff7!:  Isa(LakeHuron,{Lake,BodyOfWater})

 CPU time : 0.01

: ;;; List the bodies of water.
askwh Isa(?x, BodyOfWater)

I wonder if  p3:   Isa(x,BodyOfWater)
holds within the BS defined by context default-defaultct

I know  wff4!:  Isa({LakeOntario,LakeErie,LakeHuron},{Lake,BodyOfWater})

I know  wff5!:  Isa(StLawrenceRiver,{River,BodyOfWater})
```

```
I know  wff7!:  Isa(LakeHuron,{Lake,BodyOfWater})

((?x . StLawrenceRiver))
((?x . LakeHuron))
((?x . LakeErie))
((?x . LakeOntario))

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/reductionTrace.snepslog demonstration.
```

4 Symmetric Arguments

Above, we said that "Wherever an argument is allowed in SNePSLOG, we can put . . . a set of terms." The use of "set" here is the mathematical notion of set: order doesn't matter; and elements are either in or not in the set—there is no notion of being in the set more than once. Sets are used in SNePS in many places, and both these properties of sets are significant. We discuss and demonstrate some of the significance of this in this subsection.

Firstly, in standard logic, conjunction is a binary connective that distinguishes its two arguments. That is, $A \wedge B$ is different from $B \wedge A$. The SNePSLOG conjunction operator, `and`, however, takes a set of arguments even though it is entered as an infix operator. For example, `A and B` is the same as `B and A`:

Also, we can take advantage of sets of terms whenever we want to represent symmetric relations: those that do not distinguish the order of their arguments. For example Lake Erie and Ohio are adjacent to each other.

Not only do argument sets ignore the order of their elements, the second reduction rule of inference allows SNePS to infer a subset.

Reduction Inference (b): $\mathtt{P}(\mathtt{a}_1, \ldots, \phi, \ldots, \mathtt{a}_n) \vdash \mathtt{P}(\mathtt{a}_1, \ldots, \psi, \ldots, \mathtt{a}_n)$ if $\psi \subset \phi$.

These features are demonstrated below.

```
: demo /projects/shapiro/Sneps/SnepsIntro/symmetric.snepslog

File /projects/shapiro/Sneps/SnepsIntro/symmetric.snepslog is now the source of input.


 CPU time : 0.00

: ;;; Empty the Knowledge Base.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; (A and B) is the same as (B and A).
BettyIsDriver() and TomIsPassenger().

  wff3!:  TomIsPassenger() and BettyIsDriver()

 CPU time : 0.00
```

```
: TomIsPassenger() and BettyIsDriver().

  wff3!:  TomIsPassenger() and BettyIsDriver()

 CPU time : 0.00

: list-wffs
  wff3!:  TomIsPassenger() and BettyIsDriver()


 CPU time : 0.00

:
;;; Lake Erie and Ohio are adjacent,
Adjacent({LakeErie, Ohio}).

  wff4!:  Adjacent({Ohio,LakeErie})

 CPU time : 0.00

:
;;;    and so are Lake Erie, Pennsylvania, and New York.
Adjacent({LakeErie, Pennsylvania, NewYork}).

  wff5!:  Adjacent({NewYork,Pennsylvania,LakeErie})

 CPU time : 0.01

:
;;; What is Lake Erie adjacent to?
Adjacent({LakeErie, ?x})?

  wff7!:  Adjacent({Pennsylvania,LakeErie})
  wff6!:  Adjacent({NewYork,LakeErie})
  wff4!:  Adjacent({Ohio,LakeErie})

 CPU time : 0.00

:
;;; Are New York and Ohio adjacent?
Adjacent({NewYork, Ohio})?


 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/symmetric.snepslog demonstration.
```

5 Negation

5.1 Negated Formulas as Assertions

So far, the only non-atomic beliefs we have seen being asserted into the knowledge base have been simple conjunctions. The next simplest non-atomic formula is a negation. To represent the negation of an formula, precede it with a "~".

```
: demo /projects/shapiro/Sneps/SnepsIntro/negation.snepslog

File /projects/shapiro/Sneps/SnepsIntro/negation.snepslog is now the source of input.


 CPU time : 0.01

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; Betty drives Tom.
BettyDrivesTom().

  wff1!:  BettyDrivesTom()

 CPU time : 0.00

: ;;; Betty is the driver.
BettyIsDriver().

  wff2!:  BettyIsDriver()

 CPU time : 0.00

: ;;; Betty is not the passenger.
~BettyIsPassenger().

  wff4!:  ~BettyIsPassenger()

 CPU time : 0.00

: ;;; List the asserted beliefs
list-asserted-wffs
  wff4!:  ~BettyIsPassenger()
  wff2!:  BettyIsDriver()
  wff1!:  BettyDrivesTom()


 CPU time : 0.00

:
```

```
End of /projects/shapiro/Sneps/SnepsIntro/negation.snepslog demonstration.
```

5.2 Querying: ?, ask, askifnot, askwh, askwhnot

Now that we have negative beliefs as well as positive beliefs, we can demonstrate a larger group of query operators.

```
: demo /projects/shapiro/Sneps/SnepsIntro/asknegation.snepslog

File /projects/shapiro/Sneps/SnepsIntro/asknegation.snepslog is now the source of input.


 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; Lakes Huron, Ontario, LMichigan, Erie, and Huron are lakes.
Lake({LakeHuron, LakeOntario, LakeMichigan, LakeErie, LakeSuperior}).

  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})

 CPU time : 0.00

:
;;; Lakes Superior and Michigan feed Lake Huron.
Feeds({LakeSuperior, LakeMichigan}, LakeHuron).

  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)

 CPU time : 0.00

:
;;; Lake Ontario doesn't feed Lake Huron.
~Feeds(LakeOntario, LakeHuron).

  wff4!:  ~Feeds(LakeOntario,LakeHuron)

 CPU time : 0.00

:
;;; If you ask a question by entering a wff followed by a question mark,
;;; SNePS will give both positive and negative answers to the question:
;;; What does and doesn't feed Lake Huron?
Feeds(?x, LakeHuron)?

  wff6!:  Feeds(LakeMichigan,LakeHuron)
  wff5!:  Feeds(LakeSuperior,LakeHuron)
```

```
  wff4!:  ~Feeds(LakeOntario,LakeHuron)

 CPU time : 0.01

:
;;; If, instead, you only want positive answers, or only negative answers,
;;; use the ask command followed by the wff you want to query:
;;; What does feed Lake Huron?
ask Feeds(?x, LakeHuron)

  wff6!:  Feeds(LakeMichigan,LakeHuron)
  wff5!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

: ;;; What doesn't feed Lake Huron?
ask ~Feeds(?x, LakeHuron)

  wff4!:  ~Feeds(LakeOntario,LakeHuron)

 CPU time : 0.01

:
;;; An alternative to ask with a negated query, is:
askifnot Feeds(?x, LakeHuron)

  wff4!:  ~Feeds(LakeOntario,LakeHuron)

 CPU time : 0.00

:
;;; Can even do
askifnot ~Feeds(?x, LakeHuron)

  wff6!:  Feeds(LakeMichigan,LakeHuron)
  wff5!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.01

:
;;; For just a list of satisfying constants:
;;; What does feed Lake Huron?
askwh Feeds(?x, LakeHuron)

((?x . LakeMichigan))
((?x . LakeSuperior))

 CPU time : 0.00

: ;;; What doesn't feed Lake Huron?
askwhnot Feeds(?x, LakeHuron)
```

```
((?x . LakeOntario))

 CPU time : 0.01

:

End of /projects/shapiro/Sneps/SnepsIntro/asknegation.snepslog demonstration.
```

6 Handling Contradictions, Part I

The ability to assert negated, as well as positive propositions brings with it the possibility of a contradictory knowledge base. SNePS recognizes contradictions as soon as they become explicit, and consults the user about how to resolve the contradiction.

```
: demo /projects/shapiro/Sneps/SnepsIntro/contradictions1.snepslog

File /projects/shapiro/Sneps/SnepsIntro/contradictions1.snepslog is now the source of input


 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; Turn on inference tracing
trace inference
Tracing inference.


 CPU time : 0.00

: ;;; Load this file so that SNeBR interactions will be with the user,
;;;    even in the middle of a demo.
^^

--> (defvar *newSnebrioPath* "/projects/shapiro/Sneps/new-snebrio"
  "Location of new-snebrio.  Should use logical pathnames.")
*newSnebrioPath*
--> (cl:load *newSnebrioPath*)
; Fast loading /projects/shapiro/Sneps/new-snebrio.fasl
t
--> ^^



 CPU time : 0.02
```

```
:
;;; Lakes Huron, Ontario, Michigan, Erie, and Superior are lakes.
Lake({LakeHuron, LakeOntario, LakeMichigan, LakeErie, LakeSuperior}).

  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})

 CPU time : 0.00

:
;;; Lakes Superior and Michigan feed Lake Huron.
Feeds({LakeSuperior, LakeMichigan}, LakeHuron).

  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)

 CPU time : 0.00

:
;;; Lake Ontario doesn't feed Lake Huron.
~Feeds(LakeOntario, LakeHuron).

  wff4!:  ~Feeds(LakeOntario,LakeHuron)

 CPU time : 0.00

:
;;; Someone (mistakenly) asserts that Lake Ontario does feed Lake Huron.
Feeds(LakeOntario, LakeHuron).


  A contradiction was detected within context default-defaultct.
  The contradiction involves the proposition you want to assert:
      wff3!:  Feeds(LakeOntario,LakeHuron)

  and the previously existing proposition:
      wff4!:  ~Feeds(LakeOntario,LakeHuron)



  You have the following options:
   1. [c] to continue anyway, knowing that a contradiction is derivable;
   2. [r] to revise the inconsistent part of the context
   3. [d] to discard this contradictory new assertion from the context
  (please type c, r or d)
=><= d

  wff3:   Feeds(LakeOntario,LakeHuron)

 CPU time : 0.00

:
;;; See what's in the KB
```

```
list-wffs
  wff4!:  ~Feeds(LakeOntario,LakeHuron)
  wff3:   Feeds(LakeOntario,LakeHuron)
  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)
  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})


 CPU time : 0.00

: ;;; That shows all wffs that were ever asserted.
;;; Note that wff3 is not asserted.
;;; To show only currently asserted wffs:
list-asserted-wffs
  wff4!:  ~Feeds(LakeOntario,LakeHuron)
  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)
  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})


 CPU time : 0.00

:
;;; Lake Erie is at latitude 48N, longitude 88W.
Location(LakeErie, latLong(48N, 88W)).

  wff6!:  Location(LakeErie,latLong(48N,88W))

 CPU time : 0.01

:
;;; No, that's not right.
~Location(LakeErie, latLong(48N, 88W)).


  A contradiction was detected within context default-defaultct.
  The contradiction involves the proposition you want to assert:
      wff7!:  ~Location(LakeErie,latLong(48N,88W))

  and the previously existing proposition:
      wff6!:  Location(LakeErie,latLong(48N,88W))



  You have the following options:
   1. [c] to continue anyway, knowing that a contradiction is derivable;
   2. [r] to revise the inconsistent part of the context
   3. [d] to discard this contradictory new assertion from the context
  (please type c, r or d)
=><= r


  In order to make the context consistent you must delete at least
```

```
  one hypothesis from each of the following sets of hypotheses:
    (wff7 wff6)


 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.

An inconsistent set of hypotheses:

 1 :   wff7!:  ~Location(LakeErie,latLong(48N,88W))
          (1 supported proposition: (wff7) )

 2 :   wff6!:  Location(LakeErie,latLong(48N,88W))
          (1 supported proposition: (wff6) )


  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 2

The consistent set of hypotheses:

 1 :   wff7!:  ~Location(LakeErie,latLong(48N,88W))
          (1 supported proposition: (wff7) )


  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= q

  The following (not known to be inconsistent) set of
  hypotheses was also part of the context where the
  contradiction was derived:
    (wff4 wff2 wff1)
```

```
  Do you want to inspect or discard some of them?
=><= no

  Do you want to add a new hypothesis?  no

  wff7!:  ~Location(LakeErie,latLong(48N,88W))

 CPU time : 0.01

:
;;; See what's in the KB
list-asserted-wffs
  wff7!:  ~Location(LakeErie,latLong(48N,88W))
  wff4!:  ~Feeds(LakeOntario,LakeHuron)
  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)
  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})


 CPU time : 0.00

:
;;; Someone asserts that Lake Superior does not feed Lake Huron.
~Feeds(LakeSuperior, LakeHuron).

  wff9!:  ~Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

:
;;; The contradiction is not immediately explicit,
;;; but shows up when SNePS does reasoning.
;;; Does Lake Superior feed Lake Huron?
Feeds(LakeSuperior, LakeHuron)?

I wonder if  wff8:   Feeds(LakeSuperior,LakeHuron)
holds within the BS defined by context default-defaultct

I know it is not the case that  wff8:   Feeds(LakeSuperior,LakeHuron)

I know  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)


  A contradiction was detected within context default-defaultct.
  The contradiction involves the newly derived proposition:
      wff8!:  Feeds(LakeSuperior,LakeHuron)

  and the previously existing proposition:
      wff9!:  ~Feeds(LakeSuperior,LakeHuron)

  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)
=><= r


  In order to make the context consistent you must delete at least
  one hypothesis from each of the following sets of hypotheses:
    (wff9 wff2)


 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.

An inconsistent set of hypotheses:

 1 :   wff9!:  ~Feeds(LakeSuperior,LakeHuron)
          (1 supported proposition: (wff9) )

 2 :   wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)
          (2 supported propositions: (wff8 wff2) )


  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 1

The consistent set of hypotheses:

 1 :   wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)
          (2 supported propositions: (wff8 wff2) )


  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions
```

```
  (please type a number OR d, a, r, q or i)
=><= q

  The following (not known to be inconsistent) set of
  hypotheses was also part of the context where the
  contradiction was derived:
    (wff7 wff4 wff1)

  Do you want to inspect or discard some of them?
=><= no

  Do you want to add a new hypothesis?  no

I know  wff8!:  Feeds(LakeSuperior,LakeHuron)

  wff8!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.02

:
;;; See what's in the KB
list-asserted-wffs
  wff8!:  Feeds(LakeSuperior,LakeHuron)
  wff7!:  ~Location(LakeErie,latLong(48N,88W))
  wff4!:  ~Feeds(LakeOntario,LakeHuron)
  wff2!:  Feeds({LakeSuperior,LakeMichigan},LakeHuron)
  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})


 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/contradictions1.snepslog demonstration.
```

7 Formula-Based Inference

7.1 Simple Implication, Forward and Backward Inference

The second kind of inference in SNePS, after reduction inference, is formula-based inference. The simplest formula-based inference is simple implication, from a formula of the form A and one of the form A => B, it follows that B. This can be expressed as the rule of inference,

Implication Elimination (a): $\mathtt{A}, \mathtt{A} => \mathtt{B} \vdash \mathtt{B}$.

SNePS does not perform an inference whenever the assertions in the knowledge base warrent it, but only when it is asked to, either via forward reasoning or via backward reasoning. This is demonstrated here.

```
: demo /projects/shapiro/Sneps/SnepsIntro/implication.snepslog

File /projects/shapiro/Sneps/SnepsIntro/implication.snepslog is now the source of input.
```

```
 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.01

: ;;; Turn on tracing
trace inference
Tracing inference.


 CPU time : 0.00

:
;;; If Betty drives Tom, then Betty is the driver of the car.
BettyDrivesTom() => BettyIsDriver().

  wff3!:  BettyDrivesTom() => BettyIsDriver()

 CPU time : 0.00

:
;;; Forward inference:
;;; Betty drives Tom.  So what?
;;; Use ‘‘!’’ to fire forward inference.
BettyDrivesTom()!

Since  wff3!:  BettyDrivesTom() => BettyIsDriver()
and  wff1!:  BettyDrivesTom()
I infer  wff2:   BettyIsDriver()

  wff2!:  BettyIsDriver()
  wff1!:  BettyDrivesTom()

 CPU time : 0.00

:
;;; If Betty drives Tom, then Tom is the passenger in the car.
BettyDrivesTom() => TomIsPassenger().

  wff5!:  BettyDrivesTom() => TomIsPassenger()

 CPU time : 0.00

:
;;; Backward inference is triggered by ‘‘?’’
;;; Is Tom the passenger?
```

```
TomIsPassenger()?

I wonder if  wff4:   TomIsPassenger()
holds within the BS defined by context default-defaultct

I know  wff1!:  BettyDrivesTom()

Since  wff5!:  BettyDrivesTom() => TomIsPassenger()
and  wff1!:  BettyDrivesTom()
I infer  wff4:   TomIsPassenger()

  wff4!:  TomIsPassenger()

 CPU time : 0.01

:
;;; Inferred propositions are asserted in the knowledge base:
list-asserted-wffs

  wff5!:  BettyDrivesTom() => TomIsPassenger()
  wff4!:  TomIsPassenger()
  wff3!:  BettyDrivesTom() => BettyIsDriver()
  wff2!:  BettyIsDriver()
  wff1!:  BettyDrivesTom()


 CPU time : 0.01

:

End of /projects/shapiro/Sneps/SnepsIntro/implication.snepslog demonstration.
```

7.2 Universal Quantification, ? *vs.* ??

Implications are more useful if they can be quantified. For example, rather than

```
BettyDrivesTom() => (BettyIsDriver() and TomIsPassenger())
```

and

```
TomDrivesBetty => (TomIsDriver() and BettyIsPassinger())
```

it would be more useful to be able to say

```
all(x,y)(Drives(x,y) => (Driver(x) and Passenger(y))
```

As shown, universal quantification in SNePS is expressed

```
    all(x1, ..., xn)(A(x1, ..., xn) => B(x1, ..., xn))
```

Notice that `A(x1, ..., xn)` is the antecedent in the scope of the quantifier, and that `B(x1, ..., xn)` is the consequent. The scope of universal quantification in SNePS must always have both an antecedent and a consequent. A formula like `all(x)(Lake(x))`, without an antecedent, might not be handeled correctly by SNePS.

The scope of universal quantification in SNePS is not limited to simple implications. It may also be other non-atomic formulae that have antecedents and consequents. However, as simple implication is the only such construct we have seen so far, the rule of inference of universal elimination can be expressed

Universal Elimination (a):

```
    A(a1, ..., an),
    all(x1, ..., xn)(A(x1, ..., xn) => B(x1, ..., xn))
    ⊢ B(a1, ..., an)
```

This demonstration also introduces the "??" final punctuation, and differentiates it from the "?."

```
: demo /projects/shapiro/Sneps/SnepsIntro/quantify.snepslog

File /projects/shapiro/Sneps/SnepsIntro/quantify.snepslog is now the source of input.


 CPU time : 0.01

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

: trace inference
Tracing inference.


 CPU time : 0.00

:
;;; Lakes Huron, Ontario, Michigan, Erie, and Superior are lakes.
Lake({LakeHuron, LakeOntario, LakeMichigan, LakeErie, LakeSuperior}).

  wff1!:  Lake({LakeSuperior,LakeErie,LakeMichigan,LakeOntario,LakeHuron})

 CPU time : 0.00

:
;;; The Missouri, Mississippi, and St. Lawrence rivers are rivers.
River({MissouriRiver, MississippiRiver, StLawrenceRiver}).

  wff2!:  River({StLawrenceRiver,MississippiRiver,MissouriRiver})

 CPU time : 0.00

:
;;; There are two water systems:
;;;    Lake Superior to Lake Huron to Lake Erie to Lake Ontario.
;;;    and the Missouri river to the  Mississippi river.
Feeds(LakeSuperior, LakeHuron).

  wff3!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00
```

```
: Feeds(LakeHuron, LakeErie).

  wff4!:  Feeds(LakeHuron,LakeErie)

 CPU time : 0.01

: Feeds(LakeErie, LakeOntario).

  wff5!:  Feeds(LakeErie,LakeOntario)

 CPU time : 0.00

: Feeds(MissouriRiver, MississippiRiver).

  wff6!:  Feeds(MissouriRiver,MississippiRiver)

 CPU time : 0.00

:
;;; If x feeds y, then y is fed by x.
all(x,y)(Feeds(x,y) => FedBy(y,x)).

  wff7!:  all(y,x)(Feeds(x,y) => FedBy(y,x))

 CPU time : 0.00

:
;;; Is the Mississippi river fed by the Missouri river?
FedBy(MississippiRiver,MissouriRiver)?

I wonder if  wff8:   FedBy(MississippiRiver,MissouriRiver)
holds within the BS defined by context default-defaultct

I wonder if  wff6!:  Feeds(MissouriRiver,MississippiRiver)
holds within the BS defined by context default-defaultct

I know  wff6!:  Feeds(MissouriRiver,MississippiRiver)

Since  wff7!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
and  wff6!:  Feeds(MissouriRiver,MississippiRiver)
I infer  wff8:   FedBy(MississippiRiver,MissouriRiver)

  wff8!:  FedBy(MississippiRiver,MissouriRiver)

 CPU time : 0.01

:
;;; The terminal punctuation "??" queries the system,
;;; but only for propositions already asserted in the knowledge base.
;;; It does not trigger any inferences.
;;; What do we already know that’s fed by what?
```

```
FedBy(?x,?y)??

  wff8!:  FedBy(MississippiRiver,MissouriRiver)

 CPU time : 0.00

: ;;; Whereas the terminal punctuation "?", as we've already seen,
;;; triggers backward inference.
;;; What can we infer is fed by what?
FedBy(?x,?y)?

I wonder if  p3:   FedBy(x,y)
holds within the BS defined by context default-defaultct

I know  wff8!:  FedBy(MississippiRiver,MissouriRiver)

I wonder if  p1:   Feeds(x,y)
holds within the BS defined by context default-defaultct

Since  wff7!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
and  wff6!:  Feeds(MissouriRiver,MississippiRiver)
I infer  wff8!:  FedBy(MississippiRiver,MissouriRiver)

I know  wff3!:  Feeds(LakeSuperior,LakeHuron)

Since  wff7!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
and  wff3!:  Feeds(LakeSuperior,LakeHuron)
I infer  wff10:   FedBy(LakeHuron,LakeSuperior)

I know  wff4!:  Feeds(LakeHuron,LakeErie)

Since  wff7!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
and  wff4!:  Feeds(LakeHuron,LakeErie)
I infer  wff11:   FedBy(LakeErie,LakeHuron)

I know  wff5!:  Feeds(LakeErie,LakeOntario)

Since  wff7!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
and  wff5!:  Feeds(LakeErie,LakeOntario)
I infer  wff12:   FedBy(LakeOntario,LakeErie)

I know  wff6!:  Feeds(MissouriRiver,MississippiRiver)

  wff12!:  FedBy(LakeOntario,LakeErie)
  wff11!:  FedBy(LakeErie,LakeHuron)
  wff10!:  FedBy(LakeHuron,LakeSuperior)
  wff8!:  FedBy(MississippiRiver,MissouriRiver)

 CPU time : 0.01

:
```

```
;;; If x feeds y, then y does not feed x.
all(x,y)(Feeds(x,y) => ~Feeds(y,x)).

  wff13!:  all(y,x)(Feeds(x,y) => (~Feeds(y,x)))

 CPU time : 0.01

:
;;; What do we already know that feeds what?
Feeds(?x,?y)??

  wff3!:  Feeds(LakeSuperior,LakeHuron)
  wff4!:  Feeds(LakeHuron,LakeErie)
  wff5!:  Feeds(LakeErie,LakeOntario)
  wff6!:  Feeds(MissouriRiver,MississippiRiver)

 CPU time : 0.00

: ;;; What can we infer feeds and does not feed what?
Feeds(?x,?y)?

I wonder if  p7:   Feeds(x,y)
holds within the BS defined by context default-defaultct

I know  wff3!:  Feeds(LakeSuperior,LakeHuron)

I know  wff4!:  Feeds(LakeHuron,LakeErie)

I know  wff5!:  Feeds(LakeErie,LakeOntario)

I know  wff6!:  Feeds(MissouriRiver,MississippiRiver)

I wonder if  p4:   Feeds(x,y)
holds within the BS defined by context default-defaultct

I know  wff3!:  Feeds(LakeSuperior,LakeHuron)

Since  wff13!:  all(y,x)(Feeds(x,y) => (~Feeds(y,x)))
and  wff3!:  Feeds(LakeSuperior,LakeHuron)
I infer  wff15:   ~Feeds(LakeHuron,LakeSuperior)

I know it is not the case that  wff14:   Feeds(LakeHuron,LakeSuperior)

I know  wff4!:  Feeds(LakeHuron,LakeErie)

Since  wff13!:  all(y,x)(Feeds(x,y) => (~Feeds(y,x)))
and  wff4!:  Feeds(LakeHuron,LakeErie)
I infer  wff17:   ~Feeds(LakeErie,LakeHuron)

I know it is not the case that  wff16:   Feeds(LakeErie,LakeHuron)
```

```
I know  wff5!:  Feeds(LakeErie,LakeOntario)

Since  wff13!:  all(y,x)(Feeds(x,y) => (~Feeds(y,x)))
and  wff5!:  Feeds(LakeErie,LakeOntario)
I infer  wff19:   ~Feeds(LakeOntario,LakeErie)

I know it is not the case that  wff18:   Feeds(LakeOntario,LakeErie)

I know  wff6!:  Feeds(MissouriRiver,MississippiRiver)

Since  wff13!:  all(y,x)(Feeds(x,y) => (~Feeds(y,x)))
and  wff6!:  Feeds(MissouriRiver,MississippiRiver)
I infer  wff21:   ~Feeds(MississippiRiver,MissouriRiver)

I know it is not the case that  wff20:   Feeds(MississippiRiver,MissouriRiver)

  wff21!:  ~Feeds(MississippiRiver,MissouriRiver)
  wff19!:  ~Feeds(LakeOntario,LakeErie)
  wff17!:  ~Feeds(LakeErie,LakeHuron)
  wff15!:  ~Feeds(LakeHuron,LakeSuperior)
  wff6!:  Feeds(MissouriRiver,MississippiRiver)
  wff5!:  Feeds(LakeErie,LakeOntario)
  wff4!:  Feeds(LakeHuron,LakeErie)
  wff3!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.04

:
;;; Note that "P??" is answered only by positive instances of P,
;;; Whereas "P?" is answered by positive and negative instances of P.
Feeds(?x,?y)??

  wff3!:  Feeds(LakeSuperior,LakeHuron)
  wff4!:  Feeds(LakeHuron,LakeErie)
  wff5!:  Feeds(LakeErie,LakeOntario)
  wff6!:  Feeds(MissouriRiver,MississippiRiver)

 CPU time : 0.00

: ~Feeds(?x,?y)??

  wff15!:  ~Feeds(LakeHuron,LakeSuperior)
  wff17!:  ~Feeds(LakeErie,LakeHuron)
  wff19!:  ~Feeds(LakeOntario,LakeErie)
  wff21!:  ~Feeds(MississippiRiver,MissouriRiver)

 CPU time : 0.00

: Feeds(?x,?y)?

I wonder if  p8:   Feeds(x,y)
```

```
holds within the BS defined by context default-defaultct

I know  wff3!:  Feeds(LakeSuperior,LakeHuron)

I know  wff4!:  Feeds(LakeHuron,LakeErie)

I know  wff5!:  Feeds(LakeErie,LakeOntario)

I know  wff6!:  Feeds(MissouriRiver,MississippiRiver)

I know it is not the case that  wff14:   Feeds(LakeHuron,LakeSuperior)

I know it is not the case that  wff16:   Feeds(LakeErie,LakeHuron)

I know it is not the case that  wff18:   Feeds(LakeOntario,LakeErie)

I know it is not the case that  wff20:   Feeds(MississippiRiver,MissouriRiver)

  wff21!:  ~Feeds(MississippiRiver,MissouriRiver)
  wff19!:  ~Feeds(LakeOntario,LakeErie)
  wff17!:  ~Feeds(LakeErie,LakeHuron)
  wff15!:  ~Feeds(LakeHuron,LakeSuperior)
  wff6!:  Feeds(MissouriRiver,MississippiRiver)
  wff5!:  Feeds(LakeErie,LakeOntario)
  wff4!:  Feeds(LakeHuron,LakeErie)
  wff3!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.02

:

End of /projects/shapiro/Sneps/SnepsIntro/quantify.snepslog demonstration.
```

7.3 Existential Quantification

The existential quantifier has not been implemented in SNePS 2 (However, see §7.13). However, this is not a fatal flaw, because the existential quantifier is not needed. Note that

$$\begin{aligned} & \forall x[Parent(x) \Leftrightarrow \exists y ChildOf(y,x)] \\ \Leftrightarrow \quad & \forall x[[Parent(x) \Rightarrow \exists y ChildOf(y,x)] \wedge [\exists y ChildOf(y,x) \Rightarrow Parent(x)]] \\ \Leftrightarrow \quad & [\forall x[Parent(x) \Rightarrow \exists y ChildOf(y,x)] \wedge [\forall x[\exists y ChildOf(y,x) \Rightarrow Parent(x)]]] \\ \Leftrightarrow \quad & [\forall x[Parent(x) \Rightarrow \exists y ChildOf(y,x)] \wedge [\forall x \forall y[ChildOf(y,x) \Rightarrow Parent(x)]]] \end{aligned}$$

Moreover, we may replace

$$\forall x[Parent(x) \Rightarrow \exists y ChildOf(y,x)]$$

by

$$\forall x[Parent(x) \Rightarrow ChildOf(f(x),x)]$$

where f is a Skolem function not previously used in the knowledge base.[2] Therefore, we can represent

$$\forall x[Parent(x) \Leftrightarrow \exists y ChildOf(y,x)]$$

[2] In general, the Skolemized version of a formula is not logically equivalent to the original formula (Chang and Lee, 1973), rather the original formula is unsatisfiable if and only if the Skolemized version is. Nevertheless, given the semantics of SNePS, we believe that the replacement is acceptable.

in SNePS as

```
all(x)(Parent(x) => ChildOf(f(x),x)) and all(x,y)(ChildOf(y,x) => Parent(x))
```

For example:

```
: demo /projects/shapiro/Sneps/SnepsIntro/exist.snepslog

File /projects/shapiro/Sneps/SnepsIntro/exist.snepslog is now the source of input.


 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

:
;;; The definition of parent with the existential quantifier replaced.
all(x)(Parent(x) => ChildOf(c(x),x)) and all(x,y)(ChildOf(y,x) => Parent(x)).

  wff3!:  (all(y,x)(ChildOf(y,x) => Parent(x)))
          and (all(x)(Parent(x) => ChildOf(c(x),x)))

 CPU time : 0.00

:
;;; Betty is a parent.
Parent(Betty).

  wff4!:  Parent(Betty)

 CPU time : 0.00

:
;;; Tom is the child of Jane.
ChildOf(Tom,Jane).

  wff5!:  ChildOf(Tom,Jane)

 CPU time : 0.00

:
;;; Who are the parents?
Parent(?x)?

  wff6!:  Parent(Jane)
  wff4!:  Parent(Betty)

 CPU time : 0.01
```

```
:
;;; Who is the child of whom?
ChildOf(?x,?y)?

  wff10!:  ChildOf(c(Betty),Betty)
  wff8!:  ChildOf(c(Jane),Jane)
  wff5!:  ChildOf(Tom,Jane)

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/exist.snepslog demonstration.
```

Notice that `c(Jane)` is the child of Jane implied by the definition of `Parent`, whereas we know that `Tom` is a child of `Jane`. The two mental entities `c(Jane)` and `Tom` might be coreferential, or might not be (`Jane` might have another child).

7.4 Bi-Directional Inference

Forward and backward inference combine in SNePS to form "bi-directional inference" (Shapiro et al., 1982; Shapiro, 1987). The feature of bi-directional inference demonstrated in this section is that if a question is asked, triggering backward inference, and then new information is added with forward inference that is relevant to the question, the forward inferencing is focussed on the outstanding question, and can lead to more answers. This is effected by an organization of interacting processes called an "active connection graph" (ACG) (McKay and Shapiro, 1981; Shubin, 1981), which is built during forward and backward inference. Another use of the ACG is to retain intermediate results of the inference, so that if a question is asked for which the ACG can provide answers, it does so without repeating any of the work. To "change the subject" from the last inference(s), the ACG can be deleted with the `clear-infer` command.

```
: demo /projects/shapiro/Sneps/SnepsIntro/bidir.snepslog

File /projects/shapiro/Sneps/SnepsIntro/bidir.snepslog is now the source of input.


 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

: trace inference
Tracing inference.


 CPU time : 0.00

:
;;; The Mississippi-Missouri river basin and the St. Lawrence Seaway
;;; are part of the Atlantic Drainage Basin.
```

```
all(x)(PartOf(x, Mississippi-MissouriRiverBasin)
       => PartOf(x, AtlanticDrainageBasin)).

  wff1!:  all(x)(PartOf(x,Mississippi-MissouriRiverBasin)
                 => PartOf(x,AtlanticDrainageBasin))

 CPU time : 0.00

: all(x)(PartOf(x, StLawrenceSeaway) => PartOf(x, AtlanticDrainageBasin)).

  wff2!:  all(x)(PartOf(x,StLawrenceSeaway) => PartOf(x,AtlanticDrainageBasin))

 CPU time : 0.00

:
;;; The St. Lawrence Seaway and all canals are navigable.
all(x)(PartOf(x, StLawrenceSeaway) => Navigable(x)).

  wff3!:  all(x)(PartOf(x,StLawrenceSeaway) => Navigable(x))

 CPU time : 0.00

: all(x)(Isa(x, Canal) => Navigable(x)).

  wff4!:  all(x)(Isa(x,Canal) => Navigable(x))

 CPU time : 0.00

:
;;; A navigable entity is used for shipping.
all(x)(Navigable(x) => Function(x, Shipping)).

  wff5!:  all(x)(Navigable(x) => Function(x,Shipping))

 CPU time : 0.00

:
;;; Lake Erie is part of the St. Lawrence Seaway.  Draw inferences.
;;; Full forward inference:  all conclusions drawn.
PartOf(LakeErie, StLawrenceSeaway)!

Since  wff2!:  all(x)(PartOf(x,StLawrenceSeaway) => PartOf(x,AtlanticDrainageBasin))
and  wff6!:  PartOf(LakeErie,StLawrenceSeaway)
I infer  wff7:   PartOf(LakeErie,AtlanticDrainageBasin)

Since  wff3!:  all(x)(PartOf(x,StLawrenceSeaway) => Navigable(x))
and  wff6!:  PartOf(LakeErie,StLawrenceSeaway)
I infer  wff8:   Navigable(LakeErie)

Since  wff5!:  all(x)(Navigable(x) => Function(x,Shipping))
and  wff8!:  Navigable(LakeErie)
```

```
I infer  wff9:   Function(LakeErie,Shipping)

  wff9!:  Function(LakeErie,Shipping)
  wff8!:  Navigable(LakeErie)
  wff7!:  PartOf(LakeErie,AtlanticDrainageBasin)
  wff6!:  PartOf(LakeErie,StLawrenceSeaway)

 CPU time : 0.01

:
;;; The Erie Canal is a canal.
Isa(ErieCanal, Canal).

  wff10!:  Isa(ErieCanal,Canal)

 CPU time : 0.00

:
;;; What is used for shipping?
;;; Full backward inference: all queries asked.
Function(?x, Shipping)?

I wonder if  p11:   Function(x,Shipping)
holds within the BS defined by context default-defaultct

I know  wff9!:  Function(LakeErie,Shipping)

I wonder if  p9:   Navigable(x)
holds within the BS defined by context default-defaultct

Since  wff5!:  all(x)(Navigable(x) => Function(x,Shipping))
and  wff8!:  Navigable(LakeErie)
I infer  wff9!:  Function(LakeErie,Shipping)

I know  wff8!:  Navigable(LakeErie)

I wonder if  p5:   PartOf(x,StLawrenceSeaway)
holds within the BS defined by context default-defaultct

Since  wff3!:  all(x)(PartOf(x,StLawrenceSeaway) => Navigable(x))
and  wff6!:  PartOf(LakeErie,StLawrenceSeaway)
I infer  wff8!:  Navigable(LakeErie)

I wonder if  p7:   Isa(x,Canal)
holds within the BS defined by context default-defaultct

I know  wff6!:  PartOf(LakeErie,StLawrenceSeaway)

I know  wff10!:  Isa(ErieCanal,Canal)

Since  wff4!:  all(x)(Isa(x,Canal) => Navigable(x))
```

```
and  wff10!:  Isa(ErieCanal,Canal)
I infer  wff11:   Navigable(ErieCanal)

Since  wff5!:  all(x)(Navigable(x) => Function(x,Shipping))
and  wff11!:  Navigable(ErieCanal)
I infer  wff12:   Function(ErieCanal,Shipping)

  wff12!:  Function(ErieCanal,Shipping)
  wff9!:  Function(LakeErie,Shipping)

 CPU time : 0.02

:
;;; What is navigable?
;;; The Active Connection Graph (ACG) still exists.
;;;    Backward inference is short-circuited.
Navigable(?x)?

I wonder if  p12:   Navigable(x)
holds within the BS defined by context default-defaultct

I know  wff8!:  Navigable(LakeErie)

I know  wff11!:  Navigable(ErieCanal)

  wff11!:  Navigable(ErieCanal)
  wff8!:  Navigable(LakeErie)

 CPU time : 0.01

:
;;; Delete the ACG.
clear-infer


 CPU time : 0.00

:
;;; What is navigable?
Navigable(?x)?

I wonder if  p13:   Navigable(x)
holds within the BS defined by context default-defaultct

I know  wff8!:  Navigable(LakeErie)

I know  wff11!:  Navigable(ErieCanal)

I wonder if  p5:   PartOf(x,StLawrenceSeaway)
holds within the BS defined by context default-defaultct
```

```
I wonder if  p7:   Isa(x,Canal)
holds within the BS defined by context default-defaultct

I know  wff6!:  PartOf(LakeErie,StLawrenceSeaway)

Since  wff3!:  all(x)(PartOf(x,StLawrenceSeaway) => Navigable(x))
and  wff6!:  PartOf(LakeErie,StLawrenceSeaway)
I infer  wff8!:  Navigable(LakeErie)

I know  wff10!:  Isa(ErieCanal,Canal)

Since  wff4!:  all(x)(Isa(x,Canal) => Navigable(x))
and  wff10!:  Isa(ErieCanal,Canal)
I infer  wff11!:  Navigable(ErieCanal)

  wff11!:  Navigable(ErieCanal)
  wff8!:  Navigable(LakeErie)

 CPU time : 0.02

:
;;; Lake Ontario is part of the St. Lawrence Seaway.  Draw inferences.
;;; The ACG focuses forward inference on the active question.
;;; The conclusion that Lake Ontarion is part of the Atlantic Drainage Basin
;;; is not drawn.
PartOf(LakeOntario, StLawrenceSeaway)!

Since  wff3!:  all(x)(PartOf(x,StLawrenceSeaway) => Navigable(x))
and  wff13!:  PartOf(LakeOntario,StLawrenceSeaway)
I infer  wff14:   Navigable(LakeOntario)

  wff14!:  Navigable(LakeOntario)
  wff13!:  PartOf(LakeOntario,StLawrenceSeaway)

 CPU time : 0.00

:
;;; Check:  What do you already know that’s part of the Atlantic drainage basin?
PartOf(?x, AtlanticDrainageBasin)??

  wff7!:  PartOf(LakeErie,AtlanticDrainageBasin)

 CPU time : 0.00

:
;;; Ask fully.
;;;    What can you infer is part of the Atlantic drainage basin?
PartOf(?x, AtlanticDrainageBasin)?

I wonder if  p14:   PartOf(x,AtlanticDrainageBasin)
holds within the BS defined by context default-defaultct
```

```
I know  wff7!:  PartOf(LakeErie,AtlanticDrainageBasin)

I wonder if  p1:   PartOf(x,Mississippi-MissouriRiverBasin)
holds within the BS defined by context default-defaultct

I wonder if  p3:   PartOf(x,StLawrenceSeaway)
holds within the BS defined by context default-defaultct

Since  wff2!:  all(x)(PartOf(x,StLawrenceSeaway) => PartOf(x,AtlanticDrainageBasin))
and  wff13!:  PartOf(LakeOntario,StLawrenceSeaway)
I infer  wff15:   PartOf(LakeOntario,AtlanticDrainageBasin)

Since  wff2!:  all(x)(PartOf(x,StLawrenceSeaway) => PartOf(x,AtlanticDrainageBasin))
and  wff6!:  PartOf(LakeErie,StLawrenceSeaway)
I infer  wff7!:  PartOf(LakeErie,AtlanticDrainageBasin)

I know  wff6!:  PartOf(LakeErie,StLawrenceSeaway)

I know  wff13!:  PartOf(LakeOntario,StLawrenceSeaway)

  wff15!:  PartOf(LakeOntario,AtlanticDrainageBasin)
  wff7!:  PartOf(LakeErie,AtlanticDrainageBasin)

 CPU time : 0.01

:
```

7.5 Simple Reasoning with and

We have already seen conjunctive queries in §2.3 and §2.4. Conjunctive queries are, in fact, a use of the rule of inference of and-Introduction:

and-Introduction: A1, ..., An ⊢ A1 and ··· and An

If a conjunction is asserted in the knowledge base, any of its conjuncts may be inferred. This is the rule of inference of and-Elimination:

and-Elimination: A1 and ··· and An ⊢ A1

Although this may look like only one of the conjucts is inferred, since (A1 and ··· and An) is represented as a set, any of the conjuncts may be listed first. So the rule of inference is meant to indicate that any of the conjuncts may be inferred.

The rules of and-Introduction and and-Elimination are demonstrated for the simple two-conjunct case below.

```
: demo andReasoning.snepslog

File /projects/shapiro/Sneps/SnepsIntro/andReasoning.snepslog is now the source of input.


 CPU time : 0.00

: clearkb
Knowledge Base Cleared
```

```
 CPU time : 0.00

: untrace inference
Untracing inference.


 CPU time : 0.00

:
;;; and-Introduction
;;; Need parentheses around infixed subformula.
;;; If Betty is the driver and Tom is the passenger, then  Betty drives Tom
(BettyIsDriver() and TomIsPassenger()) => BettyDrivesTom().

  wff5!:  (TomIsPassenger() and BettyIsDriver()) => BettyDrivesTom()

 CPU time : 0.00

: ;;; Betty is the driver.
BettyIsDriver().

  wff1!:  BettyIsDriver()

 CPU time : 0.01

: ;;; Tom is the passenger.
TomIsPassenger().

  wff2!:  TomIsPassenger()

 CPU time : 0.00

: ;;; Does Betty drive Tom?
BettyDrivesTom()?

  wff4!:  BettyDrivesTom()

 CPU time : 0.00

:
;;; Not implemented with forward inference.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; If Betty is the driver and Tom is the passenger, then  Betty drives Tom
(BettyIsDriver() and TomIsPassenger()) => BettyDrivesTom().
```

```
  wff5!:  (TomIsPassenger() and BettyIsDriver()) => BettyDrivesTom()

 CPU time : 0.00

: ;;; Betty is the driver. So what?
BettyIsDriver()!

  wff1!:  BettyIsDriver()

 CPU time : 0.00

: ;;; And Tom is the passenger.So what?
TomIsPassenger()!

  wff2!:  TomIsPassenger()

 CPU time : 0.01

:
;;; But it does work to answer a pending question.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; If Betty is the driver and Tom is the passenger, then  Betty drives Tom
(BettyIsDriver() and TomIsPassenger()) => BettyDrivesTom().

  wff5!:  (TomIsPassenger() and BettyIsDriver()) => BettyDrivesTom()

 CPU time : 0.00

: ;;; Does Betty drive Tom?
BettyDrivesTom()?


 CPU time : 0.00

: ;;; Betty is the driver. So what?
BettyIsDriver()!

  wff1!:  BettyIsDriver()

 CPU time : 0.00

: ;;; And Tom is the passenger.So what?
TomIsPassenger()!

  wff4!:  BettyDrivesTom()
```

```
  wff2!:  TomIsPassenger()

 CPU time : 0.00

:
;;; and-Elimination
;;; If Betty drives Tom, then Betty is the driver and Tom is the passenger.
BettyDrivesTom() => (BettyIsDriver() and TomIsPassenger()).

  wff6!:  BettyDrivesTom() => (TomIsPassenger() and BettyIsDriver())

 CPU time : 0.01

: ;;; Betty drives Tom.
BettyDrivesTom().

  wff4!:  BettyDrivesTom()

 CPU time : 0.00

: ;;; Is Betty the driver?
BettyIsDriver()?

  wff1!:  BettyIsDriver()

 CPU time : 0.00

: ;;; Is Tom the passenger?
TomIsPassenger()?

  wff2!:  TomIsPassenger()

 CPU time : 0.00

:
;;; Again, doesn't work with forward inference.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; If Betty drives Tom, then Betty is the driver and Tom is the passenger.
BettyDrivesTom() => (BettyIsDriver() and TomIsPassenger()).

  wff5!:  BettyDrivesTom() => (TomIsPassenger() and BettyIsDriver())

 CPU time : 0.00

: ;;; Betty drives Tom.  So what?
BettyDrivesTom()!
```

```
  wff4!:  TomIsPassenger() and BettyIsDriver()
  wff1!:  BettyDrivesTom()

 CPU time : 0.00

:
;;; But does to answer a pending question.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; If Betty drives Tom, then Betty is the driver and Tom is the passenger.
BettyDrivesTom() => (BettyIsDriver() and TomIsPassenger()).

  wff5!:  BettyDrivesTom() => (TomIsPassenger() and BettyIsDriver())

 CPU time : 0.00

: ;;; Is Betty the driver?
BettyIsDriver()?


 CPU time : 0.00

: ;;; Betty drives Tom.  So what?
BettyDrivesTom()!

  wff4!:  TomIsPassenger() and BettyIsDriver()
  wff2!:  BettyIsDriver()
  wff1!:  BettyDrivesTom()

 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/andReasoning.snepslog demonstration.
```

7.6 Nested Implication

Although the formulas $(A \wedge B) \Rightarrow C$ and $A \Rightarrow (B \Rightarrow C)$ are logically equivalent, SNePS uses the wffs `(A and B) => C` and `A => (B => C)` differently when backchaining. In the first case, `A` and `B` are issued as subqueries at the same time, but in the second case, `A` is issued first, and only if it is derived is the wff `B => C` asserted and tried. The full significance of asserting the nested implication is discussed below in §7.7.

The demonstration below also shows another aspect of bi-directional inference.

```
: demo /projects/shapiro/Sneps/SnepsIntro/nested.snepslog

File /projects/shapiro/Sneps/SnepsIntro/nested.snepslog is now the source of input.
```

```
 CPU time : 0.04

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

: trace inference
Tracing inference.


 CPU time : 0.00

:
;;; All conjuncts of a conjunctive antecedent will be asked at the same time.
(BettyIsDriver() and TomIsPassenger()) => BettyDrivesTom().

  wff5!:  (TomIsPassenger() and BettyIsDriver()) => BettyDrivesTom()

 CPU time : 0.01

: BettyIsDriver().

  wff1!:  BettyIsDriver()

 CPU time : 0.00

: TomIsPassenger().

  wff2!:  TomIsPassenger()

 CPU time : 0.00

: BettyDrivesTom()?

I wonder if  wff4:   BettyDrivesTom()
holds within the BS defined by context default-defaultct

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

I know  wff2!:  TomIsPassenger()

I know  wff1!:  BettyIsDriver()

Since  wff2!:  TomIsPassenger()
and   wff1!:  BettyIsDriver()
I infer  wff3:   TomIsPassenger() and BettyIsDriver()
```

```
Since  wff5!:  (TomIsPassenger() and BettyIsDriver()) => BettyDrivesTom()
and  wff3!:  TomIsPassenger() and BettyIsDriver()
I infer  wff4:   BettyDrivesTom()

  wff4!:  BettyDrivesTom()

 CPU time : 0.01

: ;;; See what's asserted in the knowledge base:
list-asserted-wffs
  wff5!:  (TomIsPassenger() and BettyIsDriver()) => BettyDrivesTom()
  wff4!:  BettyDrivesTom()
  wff3!:  TomIsPassenger() and BettyIsDriver()
  wff2!:  TomIsPassenger()
  wff1!:  BettyIsDriver()


 CPU time : 0.00

:
;;; The outermost antecedent will be asked
;;; before the antecedent of the nested implication.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom()).

  wff5!:  BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom())

 CPU time : 0.00

: BettyIsDriver().

  wff1!:  BettyIsDriver()

 CPU time : 0.00

: TomIsPassenger().

  wff2!:  TomIsPassenger()

 CPU time : 0.00

: BettyDrivesTom()?

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

```
I know  wff1!:  BettyIsDriver()

Since  wff5!:  BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom())
and  wff1!:  BettyIsDriver()
I infer  wff4:   TomIsPassenger() => BettyDrivesTom()

I know  wff2!:  TomIsPassenger()

Since  wff4!:  TomIsPassenger() => BettyDrivesTom()
and  wff2!:  TomIsPassenger()
I infer  wff3:   BettyDrivesTom()

  wff3!:  BettyDrivesTom()

 CPU time : 0.01

: ;;; Note that the nested proposition was inferred:
list-asserted-wffs
  wff5!:  BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom())
  wff4!:  TomIsPassenger() => BettyDrivesTom()
  wff3!:  BettyDrivesTom()
  wff2!:  TomIsPassenger()
  wff1!:  BettyIsDriver()


 CPU time : 0.00

:
;;; Another aspect of bi-directional inference
;;; is that if a proposition is asserted with forward inference,
;;; and that proposition is the antecedent of an implication that is not asserted,
;;; but is the consequent of an implication,
;;; then the antecedent of that implication will be issued as a query.
clearkb
Knowledge Base Cleared


 CPU time : 0.00

: BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom()).

  wff5!:  BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom())

 CPU time : 0.00

: TomIsPassenger()!

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

  wff2!:  TomIsPassenger()
```

```
 CPU time : 0.00

: BettyIsDriver()!

Since  wff5!:  BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom())
and  wff1!:  BettyIsDriver()
I infer  wff4:   TomIsPassenger() => BettyDrivesTom()

I know  wff2!:  TomIsPassenger()

Since  wff4!:  TomIsPassenger() => BettyDrivesTom()
and  wff2!:  TomIsPassenger()
I infer  wff3:   BettyDrivesTom()

  wff4!:  TomIsPassenger() => BettyDrivesTom()
  wff3!:  BettyDrivesTom()
  wff1!:  BettyIsDriver()

 CPU time : 0.01

: list-asserted-wffs
  wff5!:  BettyIsDriver() => (TomIsPassenger() => BettyDrivesTom())
  wff4!:  TomIsPassenger() => BettyDrivesTom()
  wff3!:  BettyDrivesTom()
  wff2!:  TomIsPassenger()
  wff1!:  BettyIsDriver()


 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/nested.snepslog demonstration.
```

7.7 Lemmas: Retaining Derived Information

In mathematics, a lemma is a proposition that has been proved, is not very important in its own right, but is useful for proving theorems, because it can be used without deriving it again. SNePS stores in its knowledge base the propositions derived as a result of being asked about by its users, and also those propositions derived on the way to deriving the propositions users asked about. Like mathematical lemmas, these stored assertions can be used to shorten the reasoning needed for answering subsequent questions. So we refer to them as lemmas also.

In the following demonstration, two uses of lemmas are shown. First, if the user asks a True/False question (one that doesn't have any free variables), and the answer is already stored in the knowledge base (either as asserted or negated), then the answer is just given without trying to derive it again.

A second use of lemmas is even more interesting. It relies on the use of nested implication, and, in this demonstration, shows the ability of SNePS to deal with higher-order logic. Consider this definition of transitivity:

```
all(r)(Transitive(r) => all(x,y,z)((r(x,y) and r(y,z)) => r(x,z))).
```

Once SNePS finds out that `UpStreamOf` is transitive, it stores

```
all(x,y,z)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
```

as a lemma. This lemma is useful for reasoning about `UpStreamOf` questions, and since `Transitive(UpStreamOf)` is already stored in the knowledge base (as a lemma), SNePS doesn't have to use the more general rule to rederive it. Subsequent `UpStreamOf` questions can be answered more quickly because SNePS has, by reasoning, acquired specialized knowledge about the `UpStreamOf` relation.

This demonstration also shows that if the user knows how many answers (s)he wants to a question, SNePS can be directed to stop reasoning as soon as that many answers have been generated.

```
: demo /projects/shapiro/Sneps/SnepsIntro/lemmas.snepslog

File /projects/shapiro/Sneps/SnepsIntro/lemmas.snepslog is now the source of input.


 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

: trace inference
Tracing inference.


 CPU time : 0.00

:
;;; There are two water systems: Superior to Huron to Erie to Ontario
;;; and Missouri to Mississippi.
Feeds(LakeSuperior, LakeHuron).

  wff1!:  Feeds(LakeSuperior,LakeHuron)

 CPU time : 0.00

: Feeds(LakeHuron, LakeErie).

  wff2!:  Feeds(LakeHuron,LakeErie)

 CPU time : 0.00

: Feeds(LakeErie, LakeOntario).

  wff3!:  Feeds(LakeErie,LakeOntario)

 CPU time : 0.00

: Feeds(MissouriRiver, MississippiRiver).

  wff4!:  Feeds(MissouriRiver,MississippiRiver)

 CPU time : 0.00
```

```
:
;;; If x feeds y, then y is fed by x.
all(x,y)(Feeds(x,y) => FedBy(y,x)).

  wff5!:  all(y,x)(Feeds(x,y) => FedBy(y,x))

 CPU time : 0.00

:
;;; What is Erie fed by?
FedBy(LakeErie, ?x)?

I wonder if  p3:   FedBy(LakeErie,x)
holds within the BS defined by context default-defaultct

I wonder if  p5:   Feeds(x,LakeErie)
holds within the BS defined by context default-defaultct

I know  wff2!:  Feeds(LakeHuron,LakeErie)

Since  wff5!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
and  wff2!:  Feeds(LakeHuron,LakeErie)
I infer  wff6:   FedBy(LakeErie,LakeHuron)

  wff6!:  FedBy(LakeErie,LakeHuron)

 CPU time : 0.01

:
;;; Clear out the acg, for demo purposes.
clear-infer


 CPU time : 0.00

:
;;; See what's in the knowledge base.
list-asserted-wffs
  wff6!:  FedBy(LakeErie,LakeHuron)
  wff5!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
  wff4!:  Feeds(MissouriRiver,MississippiRiver)
  wff3!:  Feeds(LakeErie,LakeOntario)
  wff2!:  Feeds(LakeHuron,LakeErie)
  wff1!:  Feeds(LakeSuperior,LakeHuron)


 CPU time : 0.00

:
;;; Is Erie fed by Superior?
```

```
;;; Don’t need to infer it again, since it’s stored as a lemma.
FedBy(LakeErie, LakeHuron)?

I know  wff6!:  FedBy(LakeErie,LakeHuron)

  wff6!:  FedBy(LakeErie,LakeHuron)

 CPU time : 0.00

:
;;; If x feeds y, then x is upstream of y.
all(x,y)(Feeds(x,y) => UpStreamOf(x,y)).

  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))

 CPU time : 0.01

:
;;; Upstream of is transitive.
Transitive(UpStreamOf).

  wff8!:  Transitive(UpStreamOf)

 CPU time : 0.00

:
;;; Definition of transitivity.
all(r)(Transitive(r) => all(x,y,z)((r(x,y) and r(y,z)) => r(x,z))).

  wff9!:  all(r)(Transitive(r) => (all(z,y,x)((r(y,z) and r(x,y)) => r(x,z))))

 CPU time : 0.00

:
;;; Is Superior upstream of Erie?
UpStreamOf(LakeSuperior, LakeErie)?

I wonder if  wff10:   UpStreamOf(LakeSuperior,LakeErie)
holds within the BS defined by context default-defaultct

I wonder if  wff12:   Feeds(LakeSuperior,LakeErie)
holds within the BS defined by context default-defaultct

I wonder if  wff8!:  Transitive(UpStreamOf)
holds within the BS defined by context default-defaultct

I know  wff8!:  Transitive(UpStreamOf)

Since  wff9!:  all(r)(Transitive(r) => (all(z,y,x)((r(y,z) and r(x,y)) => r(x,z))))
and  wff8!:  Transitive(UpStreamOf)
I infer  wff14:   all(z,y,x)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
```

```
I wonder if  p20:   UpStreamOf(LakeSuperior,y) and UpStreamOf(y,LakeErie)
holds within the BS defined by context default-defaultct

I wonder if  wff15:   Transitive(Feeds)
holds within the BS defined by context default-defaultct

I wonder if  p18:   UpStreamOf(y,LakeErie)
holds within the BS defined by context default-defaultct

I wonder if  p19:   UpStreamOf(LakeSuperior,y)
holds within the BS defined by context default-defaultct

I wonder if  p27:   Feeds(LakeSuperior,y)
holds within the BS defined by context default-defaultct

I wonder if  p31:   Feeds(x,LakeErie)
holds within the BS defined by context default-defaultct

I wonder if  p35:   UpStreamOf(LakeSuperior,y) and UpStreamOf(y,z)
holds within the BS defined by context default-defaultct

I wonder if  p36:   UpStreamOf(y,LakeErie) and UpStreamOf(x,y)
holds within the BS defined by context default-defaultct

I wonder if  wff16:   Transitive(Transitive)
holds within the BS defined by context default-defaultct

I know  wff1!:  Feeds(LakeSuperior,LakeHuron)

Since  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
and  wff1!:  Feeds(LakeSuperior,LakeHuron)
I infer  wff17:   UpStreamOf(LakeSuperior,LakeHuron)

I know  wff2!:  Feeds(LakeHuron,LakeErie)

Since  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
and  wff2!:  Feeds(LakeHuron,LakeErie)
I infer  wff18:   UpStreamOf(LakeHuron,LakeErie)

Since  wff18!:  UpStreamOf(LakeHuron,LakeErie)
and   wff17!:  UpStreamOf(LakeSuperior,LakeHuron)
I infer  wff19:   UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)

Since  p13:   all(z,y,x)((r(y,z) and r(x,y)) => r(x,z))
and  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)
I infer  wff10:   UpStreamOf(LakeSuperior,LakeErie)

Since  p13:   all(z,y,x)((r(y,z) and r(x,y)) => r(x,z))
and  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)
I infer  wff10!:  UpStreamOf(LakeSuperior,LakeErie)
```

```
Since  p13:   all(z,y,x)((r(y,z) and r(x,y)) => r(x,z))
and  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)
I infer  wff10!:  UpStreamOf(LakeSuperior,LakeErie)

I wonder if  p15:   UpStreamOf(x,y)
holds within the BS defined by context default-defaultct

I wonder if  p19:   UpStreamOf(LakeSuperior,y)
holds within the BS defined by context default-defaultct

I wonder if  p18:   UpStreamOf(y,LakeErie)
holds within the BS defined by context default-defaultct

I wonder if  p14:   UpStreamOf(y,z)
holds within the BS defined by context default-defaultct

Since  wff17!:  UpStreamOf(LakeSuperior,LakeHuron)
and   wff18!:  UpStreamOf(LakeHuron,LakeErie)
I infer  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)

Since  wff17!:  UpStreamOf(LakeSuperior,LakeHuron)
and   wff18!:  UpStreamOf(LakeHuron,LakeErie)
I infer  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)

Since  wff14!:  all(z,y,x)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
and  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)
I infer  wff10!:  UpStreamOf(LakeSuperior,LakeErie)

Since  wff14!:  all(z,y,x)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
and  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)
I infer  wff10!:  UpStreamOf(LakeSuperior,LakeErie)

I know  wff10!:  UpStreamOf(LakeSuperior,LakeErie)

I know  wff17!:  UpStreamOf(LakeSuperior,LakeHuron)

I know  wff18!:  UpStreamOf(LakeHuron,LakeErie)

I wonder if  p6:   Feeds(x,y)
holds within the BS defined by context default-defaultct

Since  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
and  wff2!:  Feeds(LakeHuron,LakeErie)
I infer  wff18!:  UpStreamOf(LakeHuron,LakeErie)

Since  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
and  wff1!:  Feeds(LakeSuperior,LakeHuron)
I infer  wff17!:  UpStreamOf(LakeSuperior,LakeHuron)

I know  wff1!:  Feeds(LakeSuperior,LakeHuron)
```

```
I know  wff2!:  Feeds(LakeHuron,LakeErie)

I know  wff3!:  Feeds(LakeErie,LakeOntario)

Since  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
and  wff3!:  Feeds(LakeErie,LakeOntario)
I infer  wff20:   UpStreamOf(LakeErie,LakeOntario)

Since  wff10!:  UpStreamOf(LakeSuperior,LakeErie)
and   wff20!:  UpStreamOf(LakeErie,LakeOntario)
I infer  wff21:   UpStreamOf(LakeErie,LakeOntario) and UpStreamOf(LakeSuperior,LakeErie)

Since  wff14!:  all(z,y,x)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
and  wff21!:  UpStreamOf(LakeErie,LakeOntario) and UpStreamOf(LakeSuperior,LakeErie)
I infer  wff22:   UpStreamOf(LakeSuperior,LakeOntario)

I know  wff4!:  Feeds(MissouriRiver,MississippiRiver)

Since  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
and  wff4!:  Feeds(MissouriRiver,MississippiRiver)
I infer  wff23:   UpStreamOf(MissouriRiver,MississippiRiver)

  wff10!:  UpStreamOf(LakeSuperior,LakeErie)

 CPU time : 0.13

:
;;; Get rid of acg, for demo purposes.
clear-infer


 CPU time : 0.00

:
;;; See what's in the knowledge base.
list-asserted-wffs
  wff23!:  UpStreamOf(MissouriRiver,MississippiRiver)
  wff22!:  UpStreamOf(LakeSuperior,LakeOntario)
  wff21!:  UpStreamOf(LakeErie,LakeOntario) and UpStreamOf(LakeSuperior,LakeErie)
  wff20!:  UpStreamOf(LakeErie,LakeOntario)
  wff19!:  UpStreamOf(LakeHuron,LakeErie) and UpStreamOf(LakeSuperior,LakeHuron)
  wff18!:  UpStreamOf(LakeHuron,LakeErie)
  wff17!:  UpStreamOf(LakeSuperior,LakeHuron)
  wff14!:  all(z,y,x)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
  wff10!:  UpStreamOf(LakeSuperior,LakeErie)
  wff9!:  all(r)(Transitive(r) => (all(z,y,x)((r(y,z) and r(x,y)) => r(x,z))))
  wff8!:  Transitive(UpStreamOf)
  wff7!:  all(y,x)(Feeds(x,y) => UpStreamOf(x,y))
  wff6!:  FedBy(LakeErie,LakeHuron)
  wff5!:  all(y,x)(Feeds(x,y) => FedBy(y,x))
```

```
  wff4!:  Feeds(MissouriRiver,MississippiRiver)
  wff3!:  Feeds(LakeErie,LakeOntario)
  wff2!:  Feeds(LakeHuron,LakeErie)
  wff1!:  Feeds(LakeSuperior,LakeHuron)


 CPU time : 0.01

:
;;; Is Huron upstream of Ontario?
;;;    Stop as soon as you have one answer.
;;; Note that it doesn't rederive Transitive(UpStreamOf),
;;; because all(x,y,z)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
;;; is stored as a lemma.
UpStreamOf(LakeHuron, LakeOntario)? (1)

I wonder if  wff24:   UpStreamOf(LakeHuron,LakeOntario)
holds within the BS defined by context default-defaultct

I wonder if  wff26:   Feeds(LakeHuron,LakeOntario)
holds within the BS defined by context default-defaultct

I wonder if  p42:   UpStreamOf(y,LakeOntario) and UpStreamOf(LakeHuron,y)
holds within the BS defined by context default-defaultct

I wonder if  p40:   UpStreamOf(LakeHuron,y)
holds within the BS defined by context default-defaultct

I wonder if  p41:   UpStreamOf(y,LakeOntario)
holds within the BS defined by context default-defaultct

I know  wff18!:  UpStreamOf(LakeHuron,LakeErie)

I know  wff20!:  UpStreamOf(LakeErie,LakeOntario)

Since  wff18!:  UpStreamOf(LakeHuron,LakeErie)
and   wff20!:  UpStreamOf(LakeErie,LakeOntario)
I infer  wff28:   UpStreamOf(LakeErie,LakeOntario) and UpStreamOf(LakeHuron,LakeErie)

Since  wff14!:  all(z,y,x)((UpStreamOf(x,y) and UpStreamOf(y,z)) => UpStreamOf(x,z))
and  wff28!:  UpStreamOf(LakeErie,LakeOntario) and UpStreamOf(LakeHuron,LakeErie)
I infer  wff24:   UpStreamOf(LakeHuron,LakeOntario)

  wff24!:  UpStreamOf(LakeHuron,LakeOntario)

 CPU time : 0.08

:

End of /projects/shapiro/Sneps/SnepsIntro/lemmas.snepslog demonstration.
```

7.8 AndOr

Both the infix conjunction (`and`) operator and the negation (`~`) operator are actually syntactic abbreviations of the `andor` operator. Technically, `andor` is a paramaterized function from a set of propositions to a proposition. The meaning of a wff of the form `andor(i,j){P1, ..., Pn}` is the proposition that at least `i` and at most `j` of the propositions `P1, ..., Pn` are true. Certain parameter combinations yield familiar logical operators:

`andor(n,n){P1, ..., Pn}`	`P1 and ... and Pn`
`andor(1,n){P1, ..., Pn}`	`P1 or ... or Pn (inclusive)`
`andor(0,0){P}`	`~P`
`andor(0,0){P1, ..., Pn}`	`neither P1 nor ... nor Pn`
`andor(1,1){P1, ..., Pn}`	`exactly one of P1, ..., Pn`
`andor(0,n-1){P1, ..., Pn}`	`not(P1 and ... and Pn)`

The implemented rules of inference for `andor` are:

`andor(n,n)`-Elimination: `andor(n,n){P1, ..., Pn}` $\vdash$ *P1*

`andor(n,n)`-Introduction: *P1, ..., Pn* $\vdash$ `andor(n,n){P1, ..., Pn}`

`andor(0,0)`-Elimination: `andor(0,0){P1, ..., Pn}` $\vdash$ `~P1`

`andor(0,0)`-Introduction: `~P1`, ..., `~Pn` $\vdash$ `andor(0,0){P1, ..., Pn}`

`andor(i,j)` ($i < n, j > 0$) -Elimination:

1. `andor(i,j){P1, ..., Pn}`, *P1*, ..., *Pj* $\vdash$ `~Pn`, for $0 \leq i \leq j < n, 0 < j < n$,
2. `andor(i,j){`P_1`, ...,` P_n`}` `~`P_1, ..., `~`P_{n-i} $\vdash$ P_n, for $0 < i < n, i \leq j \leq n$

Some of these rules of inference are demonstrated below:

```
: demo /projects/shapiro/Sneps/SnepsIntro/andor.snepslog

File /projects/shapiro/Sneps/SnepsIntro/andor.snepslog is now the source of input.


 CPU time : 0.00

: clearkb
Knowledge Base Cleared


 CPU time : 0.00

: untrace inference
Untracing inference.


 CPU time : 0.00

:
;;; The Aral Sea, the Dead Sea,
;;;    Lake Erie, Lake Eyre, Lake Ontario,and Lake Victoria are lakes.
Lake({AralSea, DeadSea, LakeErie, LakeEyre, LakeOntario, LakeVictoria}).
```

```
  wff1!:  Lake({LakeVictoria,LakeOntario,LakeEyre,LakeErie,DeadSea,AralSea})

 CPU time : 0.00

:
;;; The Aral Sea, Lake Erie, Lake Ontario,and Lake Victoria are fresh-water lakes.
andor(4,4){Fresh(AralSea), Fresh(LakeErie),
           Fresh(LakeOntario), Fresh(LakeVictoria)}.

  wff6!:  Fresh(LakeVictoria) and Fresh(LakeOntario)
          and Fresh(LakeErie) and Fresh(AralSea)

 CPU time : 0.00

:
;;; The Dead Sea and Lake Eyre are not fresh-water lakes.
;;; E and F are not on the table.
andor(0,0){Fresh(DeadSea), Fresh(LakeEyre)}.

  wff9!:  andor(0,0){Fresh(LakeEyre),Fresh(DeadSea)}

 CPU time : 0.00

:
;;; Is Lake Erie a fresh-water lake?
Fresh(LakeErie)?

  wff3!:  Fresh(LakeErie)

 CPU time : 0.00

:
;;; Is Lake Eyre a fresh-water lake?
Fresh(LakeEyre)?

  wff10!:  ~Fresh(LakeEyre)

 CPU time : 0.00

:
;;; Every lake is either a fresh-water or a salt-water lake.
all(x)(Lake(x) => andor(1,1){Fresh(x), Salty(x)}).

  wff11!:  all(x)(Lake(x) => (andor(1,1){Salty(x),Fresh(x)}))

 CPU time : 0.00

:
;;; Is Lake Eyre a salt-water lake?
Salty(LakeEyre)?
```

```
  wff12!:  Salty(LakeEyre)

 CPU time : 0.00

:
;;; Every lake is in either America, Europe, Asia, Africa, or Australia
all(x)(Lake(x) => andor(1,1){In(x, America), In(x, Europe),
                             In(x, Asia), In(x, Africa), In(x, Australia)}).

  wff16!:  all(x)(Lake(x) => (andor(1,1){In(x,Australia),In(x,Africa),In(x,Asia),
                                         In(x,Europe),In(x,America)}))

 CPU time : 0.00

:
;;; The Aral Sea and the Dead Sea are in Asia.
In({AralSea, DeadSea}, Asia).

  wff17!:  In({DeadSea,AralSea},Asia)

 CPU time : 0.00

:
;;; Lakes Erie and Ontario are in America.
In({LakeErie, LakeOntario}, America).

  wff18!:  In({LakeOntario,LakeErie},America)

 CPU time : 0.00

:
;;; Lake Victoria is in Africa.
In(LakeVictoria, Africa).

  wff19!:  In(LakeVictoria,Africa)

 CPU time : 0.00

:
;;; Is the Dead Sea in Africa?
In(DeadSea, Africa)?

  wff21!:  ~In(DeadSea,Africa)

 CPU time : 0.03

:
;;; Lake Eyre is neither in America, nor Europe, nor Asia, nor Africa.
andor(0,0){In(LakeEyre, America), In(LakeEyre, Europe),
           In(LakeEyre, Asia), In(LakeEyre, Africa)}.
```

```
  wff36!:  andor(0,0){In(LakeEyre,Africa),In(LakeEyre,Asia),
                      In(LakeEyre,Europe),In(LakeEyre,America)}

 CPU time : 0.00

:
;;; The continents are America, Europe, Asia, Africa, and Australia
Isa({America, Europe, Asia, Africa, Australia}, Continent).

  wff37!:  Isa({Australia,Africa,Asia,Europe,America},Continent)

 CPU time : 0.00

:
;;; What continent is Lake Eyre in?
Isa(?c, Continent) and In(LakeEyre, ?c)?

  wff58!:  Isa(Australia,Continent) and In(LakeEyre,Australia)
  wff56!:  andor(0,1){Isa(America,Continent),In(LakeEyre,America)}
  wff54!:  andor(0,1){Isa(Africa,Continent),In(LakeEyre,Africa)}
  wff52!:  andor(0,1){Isa(Europe,Continent),In(LakeEyre,Europe)}
  wff50!:  andor(0,1){Isa(Asia,Continent),In(LakeEyre,Asia)}

 CPU time : 0.11

:
;;; But America, Africa, Europe, and Asia are continents!
andor(4,4){Isa(America,Continent), Isa(Africa,Continent),
           Isa(Europe,Continent), Isa(Asia,Continent)}!

  wff59!:  Isa(America,Continent) and Isa(Europe,Continent)
           and Isa(Asia,Continent) and Isa(Africa,Continent)
  wff48!:  Isa(America,Continent)
  wff47!:  Isa(Europe,Continent)
  wff46!:  Isa(Asia,Continent)
  wff45!:  Isa(Africa,Continent)
  wff41!:  ~In(LakeEyre,Africa)
  wff40!:  ~In(LakeEyre,Asia)
  wff39!:  ~In(LakeEyre,Europe)
  wff38!:  ~In(LakeEyre,America)

 CPU time : 0.03

:

End of /projects/shapiro/Sneps/SnepsIntro/andor.snepslog demonstration.
```

7.9 Thresh

Demo: `/projects/shapiro/Sneps/SnepsIntro/thresh.snepslog`

7.10 Or-Entailment

Demo: `/projects/shapiro/Sneps/SnepsIntro/orEntailment.snepslog`

7.11 And-Entailment

Demo: `/projects/shapiro/Sneps/SnepsIntro/andEntailment.snepslog` illustrates:

- three ways of expressing conjoined antecedents;
- deleting a wff from the current context;
- the ATMS disbelieving implications derived from disbelieved hypotheses;
- the triggering of backward inference from forward inference;
- and the Unique Variable Binding Rule (UVBR).

7.12 Numerical Entailment

7.13 The Numerical Quantifier

Demo: `/projects/shapiro/Sneps/SnepsIntro/numQuant.snepslog`

7.14 Hypotheses *vs.* Derived Propositions

8 Contexts

In SNePS 2, an extensional context is simply a set of hypotheses. An intensional context is a structure with a name and an extensional context. At any point, one intensional context is the current context. Newly asserted wffs (hypotheses) are put into the current context. (Actually the extensional context of the current context is changed to be the old one plus the singleton set of the new hypothesis.) A wff is considered to be asserted just in case one of its origin sets is a subset of the current context. The current belief space is the set of currently asserted wffs.

In the following demonstration, three contexts are established: the context of Stu's schedule, the context of Tony's schedule, and the union of the two. Reasoning within Stu's schedule, the faculty meeting is scheduled for the afternoon. Reasoning within Tony's schedule, the faculty meeting is scheduled for the morning. Neither of these two contexts is problematic. However, the union context is found to be contradictory, because in it, the faculty meeting is scheduled both for the morning and for the afternoon.

```
: demo /projects/shapiro/Sneps/SnepsIntro/contexts.snepslog

File /projects/shapiro/Sneps/SnepsIntro/contexts.snepslog is now the source of input.


 CPU time : 0.00

: ;;; Examples from
;;;    Joao P. Martins and Stuart C. Shapiro
;;;    Reasoning in Multiple Belief Spaces
;;;    Proc. IJCAI-83

;;; Make a knowledge base.
;;; Then create two subcontexts.
;;; Reason in each successfully.
```

```
;;; Go back to the original context.
;;; It is now known to be inconsistent.

clearkb
Knowledge Base Cleared


 CPU time : 0.00

: ;;; General Information about meetings
;;;
;;; Every meeting is either in the morning or in the afternoon.
all(x)(Meeting(x) => andor(1,1){time(x, morning), time(x, afternoon)}).

  wff1!:  all(x)(Meeting(x) => (andor(1,1){time(x,afternoon),time(x,morning)}))

 CPU time : 0.01

: ;;; At most one meeting is in the morning.
nexists(_,1,_)(x)(Meeting(x): time(x, morning)).

  wff2!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,morning)}]

 CPU time : 0.00

: ;;; At most one meeting is in the afternoon.
nexists(_,1,_)(x)(Meeting(x): time(x, afternoon)).

  wff3!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,afternoon)}]

 CPU time : 0.00

: ;;;
;;; Information for all faculty
;;;
;;; The faculty meeting is a meeting.
Meeting(facultyMeeting).

  wff4!:  Meeting(facultyMeeting)

 CPU time : 0.00

: ;;;
;;; Stu’s information
;;;
;;; The seminar is a meeting.
Meeting(seminar).

  wff5!:  Meeting(seminar)

 CPU time : 0.00
```

```
: ;;; The seminar is held in the morning.
time(seminar, morning).

  wff6!:  time(seminar,morning)

 CPU time : 0.00

: ;;;
;;; Tony's information
;;;
;;; The tennis game is a meeting.
Meeting(tennisGame).

  wff7!:  Meeting(tennisGame)

 CPU time : 0.00

: ;;; The tennis game is in the afternoon.
time(tennisGame, afternoon).

  wff8!:  time(tennisGame,afternoon)

 CPU time : 0.00

: ;;;
;;; Describe the current context
describe-context

((assertions (wff8 wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (default-defaultct)) (kinconsistent nil))

 CPU time : 0.00

: ;;; What wffs are asserted in the current context?
list-asserted-wffs
  wff8!:  time(tennisGame,afternoon)
  wff7!:  Meeting(tennisGame)
  wff6!:  time(seminar,morning)
  wff5!:  Meeting(seminar)
  wff4!:  Meeting(facultyMeeting)
  wff3!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,afternoon)}]
  wff2!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,morning)}]
  wff1!:  all(x)(Meeting(x) => (andor(1,1){time(x,afternoon),time(x,morning)}))


 CPU time : 0.01

: ;;; Change the current context to reflect Stu's schedule.
set-context StuSchedule {wff1,wff2,wff3,wff4,wff5,wff6}
```

```
((assertions (wff6 wff5 wff4 wff3 wff2 wff1)) (named (StuSchedule))
 (kinconsistent nil))

 CPU time : 0.00

: set-default-context StuSchedule

((assertions (wff6 wff5 wff4 wff3 wff2 wff1)) (named (StuSchedule))
 (kinconsistent nil))

 CPU time : 0.00

: ;;; What are the times of all meetings?
time(?x,?t)?

  wff15!:  ~time(seminar,afternoon)
  wff14!:  ~time(facultyMeeting,morning)
  wff9!:  time(facultyMeeting,afternoon)
  wff6!:  time(seminar,morning)

 CPU time : 0.04

: ;;; What wffs are asserted in the context of Stu's schedule?
list-asserted-wffs
  wff15!:  ~time(seminar,afternoon)
  wff14!:  ~time(facultyMeeting,morning)
  wff13!:  andor(1,1){time(seminar,afternoon),time(seminar,morning)}
  wff11!:  andor(1,1){time(facultyMeeting,morning),time(facultyMeeting,afternoon)}
  wff9!:  time(facultyMeeting,afternoon)
  wff6!:  time(seminar,morning)
  wff5!:  Meeting(seminar)
  wff4!:  Meeting(facultyMeeting)
  wff3!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,afternoon)}]
  wff2!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,morning)}]
  wff1!:  all(x)(Meeting(x) => (andor(1,1){time(x,afternoon),time(x,morning)}))


 CPU time : 0.01

: ;;; Clear the acg from reasoning in the context of Stu's schedule.
clear-infer


 CPU time : 0.00

: ;;; Change the current context to reflect Tony's schedule.
set-context TonySchedule {wff1,wff2,wff3,wff4,wff7,wff8}

((assertions (wff8 wff7 wff4 wff3 wff2 wff1)) (named (TonySchedule))
 (kinconsistent nil))
```

```
 CPU time : 0.00

: set-default-context TonySchedule

((assertions (wff8 wff7 wff4 wff3 wff2 wff1)) (named (TonySchedule))
 (kinconsistent nil))


 CPU time : 0.00

: ;;; What are the times of all meetings?
time(?x,?t)?

  wff19!:  ~time(tennisGame,morning)
  wff16!:  ~time(facultyMeeting,afternoon)
  wff10!:  time(facultyMeeting,morning)
  wff8!:  time(tennisGame,afternoon)

 CPU time : 0.04

: ;;; What wffs are asserted in the context of Tony's schedule?
list-asserted-wffs
  wff19!:  ~time(tennisGame,morning)
  wff18!:  andor(1,1){time(tennisGame,morning),time(tennisGame,afternoon)}
  wff16!:  ~time(facultyMeeting,afternoon)
  wff11!:  andor(1,1){time(facultyMeeting,morning),time(facultyMeeting,afternoon)}
  wff10!:  time(facultyMeeting,morning)
  wff8!:  time(tennisGame,afternoon)
  wff7!:  Meeting(tennisGame)
  wff4!:  Meeting(facultyMeeting)
  wff3!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,afternoon)}]
  wff2!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,morning)}]
  wff1!:  all(x)(Meeting(x) => (andor(1,1){time(x,afternoon),time(x,morning)}))


 CPU time : 0.01

: ;;; List the wffs that have been asserted in any context.
list-wffs
  wff19!:  ~time(tennisGame,morning)
  wff18!:  andor(1,1){time(tennisGame,morning),time(tennisGame,afternoon)}
  wff16!:  ~time(facultyMeeting,afternoon)
  wff15:   ~time(seminar,afternoon)
  wff14:   ~time(facultyMeeting,morning)
  wff13:   andor(1,1){time(seminar,afternoon),time(seminar,morning)}
  wff11!:  andor(1,1){time(facultyMeeting,morning),time(facultyMeeting,afternoon)}
  wff10!:  time(facultyMeeting,morning)
  wff9:   time(facultyMeeting,afternoon)
  wff8!:  time(tennisGame,afternoon)
  wff7!:  Meeting(tennisGame)
  wff6:   time(seminar,morning)
```

```
  wff5:   Meeting(seminar)
  wff4!:  Meeting(facultyMeeting)
  wff3!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,afternoon)}]
  wff2!:  nexists(_,1,_)(x)[{Meeting(x)}:{time(x,morning)}]
  wff1!:  all(x)(Meeting(x) => (andor(1,1){time(x,afternoon),time(x,morning)}))


 CPU time : 0.01

: ;;; Change the current context to reflect everyone's schedule.
set-default-context default-defaultct

((assertions (wff8 wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (default-defaultct)) (kinconsistent nil))


 CPU time : 0.00

: ;;; When is the seminar?
time(seminar, ?t)?

  wff15!:  ~time(seminar,afternoon)
  wff6!:  time(seminar,morning)

 CPU time : 0.01

: ;;; When is the tennis game?
time(tennisGame, ?t)?

  wff19!:  ~time(tennisGame,morning)
  wff8!:  time(tennisGame,afternoon)

 CPU time : 0.00

: ;;; When is the faculty meeting?
time(facultyMeeting, ?t)?


  A contradiction was detected within context default-defaultct.
  The contradiction involves the newly derived proposition:
      wff9!:  time(facultyMeeting,afternoon)

  and the previously existing proposition:
      wff16!:  ~time(facultyMeeting,afternoon)

  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)
=><= ;;; Drop the question.
d


 CPU time : 0.02

: ;;; Describe the contexts of Stu's and of Tony's schedules.
describe-context StuSchedule

((assertions (wff6 wff5 wff4 wff3 wff2 wff1)) (named (StuSchedule))
 (kinconsistent nil))


 CPU time : 0.00

: describe-context TonySchedule

((assertions (wff8 wff7 wff4 wff3 wff2 wff1)) (named (TonySchedule))
 (kinconsistent nil))


 CPU time : 0.00

: ;;; Describe the current context of everyone's schedule.
;;; Note that it is known to be inconsistent.
describe-context

((assertions (wff8 wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (default-defaultct)) (kinconsistent t))


 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/contexts.snepslog demonstration.
```

9 SNeBR: Handling Contradictions, Part II

10 Nodes, Case Frames, and Cablesets

10.1 show

Demo: `/projects/shapiro/Sneps/SnepsIntro/nodes.snepslog`

10.2 Atomic and Molecular Nodes

See the examples in the diagrams produced by `show`.

10.3 Case Frames and Cablesets

See the examples in the diagrams produced by `show`.

11 Mode 3 and Path-Based Inference

11.1 Mode 3 and `define-frame`

Demo: `/projects/shapiro/Sneps/SnepsIntro/mode3.snepslog`

11.2 Path-Based Inference

```
: demo /projects/shapiro/Sneps/SnepsIntro/pb-inf.snepslog

File /projects/shapiro/Sneps/SnepsIntro/pb-inf.snepslog is now the source of input.


 CPU time : 0.00

: untrace inference
Untracing inference.


 CPU time : 0.00

: normal


 CPU time : 0.00

: set-mode-3

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

achieve(x1) will be represented by {<action, achieve>, <object1, x1>}
ActPlan(x1, x2) will be represented by {<act, x1>, <plan, x2>}
believe(x1) will be represented by {<action, believe>, <object1, x1>}
disbelieve(x1) will be represented by {<action, disbelieve>, <object1, x1>}
adopt(x1) will be represented by {<action, adopt>, <object1, x1>}
unadopt(x1) will be represented by {<action, unadopt>, <object1, x1>}
do-all(x1) will be represented by {<action, do-all>, <object1, x1>}
do-one(x1) will be represented by {<action, do-one>, <object1, x1>}
Effect(x1, x2) will be represented by {<act, x1>, <effect, x2>}
else(x1) will be represented by {<else, x1>}
GoalPlan(x1, x2) will be represented by {<goal, x1>, <plan, x2>}
if(x1, x2) will be represented by {<condition, x1>, <then, x2>}
ifdo(x1, x2) will be represented by {<if, x1>, <do, x2>}
Precondition(x1, x2) will be represented by {<act, x1>, <precondition, x2>}
snif(x1) will be represented by {<action, snif>, <object1, x1>}
sniterate(x1) will be represented by {<action, sniterate>, <object1, x1>}
```

```
snsequence(x1, x2) will be represented by
                    {<action, snsequence>, <object1, x1>, <object2, x2>}
whendo(x1, x2) will be represented by {<when, x1>, <do, x2>}
wheneverdo(x1, x2) will be represented by {<whenever, x1>, <do, x2>}
withall(x1, x2, x3, x4) will be represented by
       {<action, withall>, <vars, x1>, <suchthat, x2>, <do, x3>, <else, x4>}
withsome(x1, x2, x3, x4) will be represented by
        {<action, withsome>, <vars, x1>, <suchthat, x2>, <do, x3>, <else, x4>}


 CPU time : 0.03

:
;;; Isa(x, y): x is a member of the class y.
define-frame Isa (nil member class) "[member] is a member of the class [class]"
Isa(x1, x2) will be represented by {<member, x1>, <class, x2>}


 CPU time : 0.00

: ;;; Ako(x, y): class x is a subclass of class y.
define-frame Ako (nil subclass superclass) "[subclass] is a kind of [superclass]"
Ako(x1, x2) will be represented by {<subclass, x1>, <superclass, x2>}


 CPU time : 0.00

:
;;; Lakes Huron, Ontario, Michigan, Erie, and Superior are Lakes.
Isa({Huron, LOntario, LMichigan, Erie, Superior}, Lake).

  wff1!:  Isa({Superior,Erie,LMichigan,LOntario,Huron},Lake)

 CPU time : 0.00

:
;;; The St. Lawrence is a River.
Isa(StLawrence, River).

  wff2!:  Isa(StLawrence,River)

 CPU time : 0.00

:
;;; Lakes and Rivers are bodies of water.
Ako({Lake, River}, BodyOfWater).

  wff3!:  Ako({River,Lake},BodyOfWater)

 CPU time : 0.00
```

```
:
;;; bodies of water are geographical entities.
Ako(BodyOfWater, GeographicalEntity).

  wff4!:  Ako(BodyOfWater,GeographicalEntity)

 CPU time : 0.01

:
;;; Display the KB.
show


 CPU time : 0.12

:
;;; What is Erie?
askwh Isa(Erie, ?x)

((?x . Lake))

 CPU time : 0.00

:
;;; Define a path-based inference rule for class
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-)


 CPU time : 0.00

:
;;; List the classes of Erie.
askwh Isa(Erie, ?x)

((?x . GeographicalEntity))
((?x . BodyOfWater))
((?x . Lake))

 CPU time : 0.01

:
;;; What is Erie?
Isa(Erie, ?x)?

  wff7!:  Isa(Erie,GeographicalEntity)
  wff6!:  Isa(Erie,BodyOfWater)
```

```
  wff5!:  Isa(Erie,Lake)

 CPU time : 0.00

:
;;; Ask for subclasses of geographical entities.
Ako(?x, GeographicalEntity)?

  wff4!:  Ako(BodyOfWater,GeographicalEntity)

 CPU time : 0.00

:
;;; Define path-based inference rule for subclass.
define-path subclass (compose subclass
                              (kstar (compose superclass- subclass)))
subclass implied by the path (compose subclass
                              (kstar (compose superclass- subclass)))
subclass- implied by the path (compose (kstar (compose subclass- superclass))
                               subclass-)


 CPU time : 0.00

:
;;; Ask again.
Ako(?x, GeographicalEntity)?

  wff9!:  Ako(River,GeographicalEntity)
  wff8!:  Ako(Lake,GeographicalEntity)
  wff4!:  Ako(BodyOfWater,GeographicalEntity)

 CPU time : 0.00

:
;;; A pond is a body of water.
Ako(Pond, BodyOfWater).

  wff10!:  Ako(Pond,BodyOfWater)

 CPU time : 0.00

:
;;; What is a pond?
askwh Ako(Pond, ?x)

((?x . BodyOfWater))
((?x . GeographicalEntity))

 CPU time : 0.00
```

```
:
;;; States are not bodies of water.
~Ako(State, BodyOfWater).

  wff13!:  ~Ako(State,BodyOfWater)

 CPU time : 0.00

:
;;; What are states?
Ako(State, ?x)?

  wff14!:  Ako(State,GeographicalEntity)
  wff13!:  ~Ako(State,BodyOfWater)

 CPU time : 0.00

:
;;; See why that happened.
show {wff4, wff13}


 CPU time : 0.06

:
;;; Redo with better path-based inference rules.
set-mode-3

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

achieve(x1) will be represented by {<action, achieve>, <object1, x1>}
ActPlan(x1, x2) will be represented by {<act, x1>, <plan, x2>}
believe(x1) will be represented by {<action, believe>, <object1, x1>}
disbelieve(x1) will be represented by {<action, disbelieve>, <object1, x1>}
adopt(x1) will be represented by {<action, adopt>, <object1, x1>}
unadopt(x1) will be represented by {<action, unadopt>, <object1, x1>}
do-all(x1) will be represented by {<action, do-all>, <object1, x1>}
do-one(x1) will be represented by {<action, do-one>, <object1, x1>}
Effect(x1, x2) will be represented by {<act, x1>, <effect, x2>}
else(x1) will be represented by {<else, x1>}
GoalPlan(x1, x2) will be represented by {<goal, x1>, <plan, x2>}
if(x1, x2) will be represented by {<condition, x1>, <then, x2>}
ifdo(x1, x2) will be represented by {<if, x1>, <do, x2>}
Precondition(x1, x2) will be represented by {<act, x1>, <precondition, x2>}
snif(x1) will be represented by {<action, snif>, <object1, x1>}
sniterate(x1) will be represented by {<action, sniterate>, <object1, x1>}
snsequence(x1, x2) will be represented by
                   {<action, snsequence>, <object1, x1>, <object2, x2>}
whendo(x1, x2) will be represented by {<when, x1>, <do, x2>}
```

```
whenever do(x1, x2) will be represented by {<whenever, x1>, <do, x2>}
withall(x1, x2, x3, x4) will be represented by
       {<action, withall>, <vars, x1>, <suchthat, x2>, <do, x3>, <else, x4>}
withsome(x1, x2, x3, x4) will be represented by
        {<action, withsome>, <vars, x1>, <suchthat, x2>, <do, x3>, <else, x4>}


 CPU time : 0.02

:
;;; Isa(x, y): x is a member of the class y.
define-frame Isa (nil member class) "[member] is a member of the class [class]"
Isa(x1, x2) will be represented by {<member, x1>, <class, x2>}


 CPU time : 0.01

: ;;; Ako(x, y): class x is a subclass of class y.
define-frame Ako (nil subclass superclass) "[subclass] is a kind of [superclass]"
Ako(x1, x2) will be represented by {<subclass, x1>, <superclass, x2>}


 CPU time : 0.01

:
;;; Lakes Huron, Ontario, Michigan, Erie, and Superior are Lakes.
Isa({Huron, LOntario, LMichigan, Erie, Superior}, Lake).

  wff1!:  Isa({Superior,Erie,LMichigan,LOntario,Huron},Lake)

 CPU time : 0.00

:
;;; The St. Lawrence is a River.
Isa(StLawrence, River).

  wff2!:  Isa(StLawrence,River)

 CPU time : 0.00

:
;;; Lakes and Rivers are bodies of water.
Ako({Lake, River}, BodyOfWater).

  wff3!:  Ako({River,Lake},BodyOfWater)

 CPU time : 0.00

:
;;; bodies of water are geographical entities.
Ako(BodyOfWater, GeographicalEntity).
```

```
  wff4!:  Ako(BodyOfWater,GeographicalEntity)

 CPU time : 0.00

:
;;; Define a path-based inference rule for class
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-)


 CPU time : 0.00

:
;;; What is Erie?
askwh Isa(Erie, ?x)

((?x . GeographicalEntity))
((?x . BodyOfWater))
((?x . Lake))

 CPU time : 0.00

:
;;; Define a better path-based inference rule for subclass.
define-path subclass (compose subclass
                              (kstar (compose superclass- ! subclass)))
subclass implied by the path (compose subclass
                              (kstar (compose superclass- ! subclass)))
subclass- implied by the path (compose
                               (kstar (compose subclass- ! superclass))
                               subclass-)


 CPU time : 0.00

:
;;; Ask for subclasses of geographical entities.
Ako(?x, GeographicalEntity)?

  wff9!:  Ako(River,GeographicalEntity)
  wff8!:  Ako(Lake,GeographicalEntity)
  wff4!:  Ako(BodyOfWater,GeographicalEntity)

 CPU time : 0.01

:
```

```
;;; A pond is a body of water.
Ako(Pond, BodyOfWater).

  wff10!:  Ako(Pond,BodyOfWater)

 CPU time : 0.00

:
;;; What is a pond?
askwh Ako(Pond, ?x)

((?x . BodyOfWater))
((?x . GeographicalEntity))

 CPU time : 0.00

:
;;; States are not bodies of water.
~Ako(State, BodyOfWater).

  wff13!:  ~Ako(State,BodyOfWater)

 CPU time : 0.00

:
;;; What are states?
Ako(State, ?x)?

  wff13!:  ~Ako(State,BodyOfWater)

 CPU time : 0.00

:
;;; Is a State a lake?
Ako(State, Lake)?


 CPU time : 0.01

:
;;; Display more.
show {wff3, wff4, wff13}


 CPU time : 0.08

:
;;; An even better path-based rule for superclass
define-path superclass (or
                        ;; The superclass relation goes up the class hierarchy,
                        ;; If negative, it goes down the class hierarchy.
```

```
                        (compose ! superclass
                                 (kstar (compose subclass- ! superclass)))
                        (domain-restrict ((compose arg- ! max) 0)
                                         (compose superclass
                                                  (kstar
                                                   (compose superclass- !
                                                            subclass)))))
superclass implied by the path (or (compose ! superclass
                                    (kstar (compose subclass- ! superclass)))
                                   (domain-restrict ((compose arg- ! max) 0)
                                    (compose superclass
                                     (kstar
                                      (compose superclass- ! subclass)))))
superclass- implied by the path (or (compose
                                     (kstar (compose superclass- ! subclass))
                                     superclass- !)
                                    (range-restrict
                                     (compose
                                      (kstar (compose subclass- ! superclass))
                                      superclass-)
                                     ((compose arg- ! max) 0)))


 CPU time : 0.00

:
;;; Now, is a State a lake?
Ako(State, Lake)?

  wff16!:  ~Ako(State,Lake)

 CPU time : 0.00

:
;;; What is what?
Ako(?x, ?y)?

  wff22!:  ~Ako(State,River)
  wff20!:  ~Ako(State,Pond)
  wff18!:  Ako(River,BodyOfWater)
  wff17!:  Ako(Lake,BodyOfWater)
  wff16!:  ~Ako(State,Lake)
  wff13!:  ~Ako(State,BodyOfWater)
  wff11!:  Ako(Pond,GeographicalEntity)
  wff10!:  Ako(Pond,BodyOfWater)
  wff9!:  Ako(River,GeographicalEntity)
  wff8!:  Ako(Lake,GeographicalEntity)
  wff4!:  Ako(BodyOfWater,GeographicalEntity)

 CPU time : 0.05
```

```
:
;;; Navigable(x): Entities of class x are navigable.
define-frame Navigable(property entities) "Entities of class [entities] are [property]"
Navigable(x1) will be represented by {<property, Navigable>, <entities, x1>}


 CPU time : 0.00

:
;;; If a class of entities are navigable, then they are bodies of water.
all(x)(Navigable(x) => Ako(x,BodyOfWater)).

  wff23!:  all(x)(Navigable(x) => Ako(x,BodyOfWater))

 CPU time : 0.01

:
;;; Oceans are navigable.
Navigable(Ocean).

  wff24!:  Navigable(Ocean)

 CPU time : 0.00

:
;;; Make sure the acg is cleared
clear-infer


 CPU time : 0.00

:
;;; Are oceans bodies of water?
Ako(Ocean, BodyOfWater)?


 CPU time : 0.01

:
;;; Look at the network
show {wff23,wff24,wff25}


 CPU time : 0.08

:
;;; Final version of the path for superclass
define-path superclass (or
                        superclass
                        ;; The superclass relation goes up the class hierarchy,
                        ;; If negative, it goes down the class hierarchy.
```

```
                        (compose ! superclass
                                 (kstar (compose subclass- ! superclass)))
                        (domain-restrict ((compose arg- ! max) 0)
                                         (compose superclass
                                                  (kstar
                                                   (compose superclass- !
                                                            subclass)))))

superclass already has the path definition:

 (or (compose ! snepslog::superclass
      (kstar (compose subclass- ! snepslog::superclass)))
     (domain-restrict ((compose arg- ! max) 0)
      (compose snepslog::superclass
       (kstar (compose superclass- ! snepslog::subclass)))))

Do you really want to redefine it? yes
superclass implied by the path (or superclass
                                   (compose ! superclass
                                    (kstar (compose subclass- ! superclass)))
                                   (domain-restrict ((compose arg- ! max) 0)
                                    (compose superclass
                                     (kstar
                                      (compose superclass- ! subclass)))))
superclass- implied by the path (or superclass-
                                    (compose
                                     (kstar (compose superclass- ! subclass))
                                     superclass- !)
                                    (range-restrict
                                     (compose
                                      (kstar (compose subclass- ! superclass))
                                      superclass-)
                                     ((compose arg- ! max) 0)))


 CPU time : 0.00

:
;;; Clear the acg,
clear-infer


 CPU time : 0.00

:
;;; and ask again.
;;; Are oceans bodies of water?
Ako(Ocean, BodyOfWater)?

  wff26!:  Ako(Ocean,BodyOfWater)
```

```
 CPU time : 0.00

:

End of /projects/shapiro/Sneps/SnepsIntro/pb-inf.snepslog demonstration.

 CPU time : 0.62
```

12 SNeRE: The SNePS Acting Language

References

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbol Logic and Mechanical Theorem Proving*. Academic Press, New York.

McKay, D. P. and Shapiro, S. C. (1981). Using active connection graphs for reasoning with recursive rules. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 368–374. Morgan Kaufmann, San Mateo, CA.

Shapiro, S. C. (1987). Processing, bottom-up and top-down. In Shapiro, S. C., editor, *Encyclopedia of Artificial Intelligence*, pages 779–785. John Wiley & Sons, New York. Reprinted in Second Edition, 1992, pages 1229–1234.

Shapiro, S. C., Martins, J. P., and McKay, D. P. (1982). Bi-directional inference. In *Proceedings of the Fourth Annual Meeting of the Cognitive Science Society*, pages 90–93, Ann Arbor, MI.

Shapiro, S. C. and The SNePS Implementation Group (2007). *SNePS 2.7 User's Manual*. Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY. Available as `http://www.cse.buffalo.edu/sneps/Manuals/manual27.pdf`.

Shubin, H. (1981). Inference and control in multiprocessing environments. Technical Report 186, Department of Computer Science, SUNY at Buffalo.

Appendix A: Rules of Inference

This appendix summarizes the rules of inference. For a more detailed description, including which ones operate during forward-, backward-, and forward-in-backward chaining, see (Shapiro and The SNePS Implementation Group, 2007, §6.4)

Reduction Inference (a): $\mathtt{P}(\mathtt{a_1}, \ldots, \{\ldots, \mathtt{a_{ij}}, \ldots\}, \ldots, \mathtt{a_n}) \vdash \mathtt{P}(\mathtt{a_1}, \ldots, \mathtt{a_{ij}}, \ldots, \mathtt{a_n})$

Reduction Inference (b): $\mathtt{P}(\mathtt{a_1}, \ldots, \phi, \ldots, \mathtt{a_n}) \vdash \mathtt{P}(\mathtt{a_1}, \ldots, \psi, \ldots, \mathtt{a_n})$ if $\psi \subset \phi$.

and-Introduction: $\mathtt{A1}, ..., \mathtt{An} \vdash \mathtt{A1\ and\ \cdots\ and\ An}$

and-Elimination: $\mathtt{A1\ and\ \cdots\ and\ An} \vdash \mathtt{A1}$

andor(*n*,*n*)-Elimination: $\mathtt{andor}(n,n)\{P1, \ldots, Pn\} \vdash P1$

andor(*n*,*n*)-Introduction: $P1, \ldots, Pn \vdash \mathtt{andor}(n,n)\{P1, \ldots, Pn\}$

andor(0,0)-Elimination: $\mathtt{andor}(0,0)\{P1, \ldots, Pn\} \vdash \tilde{}P1$

andor(0,0)-Introduction: $\tilde{}P1, \ldots, \tilde{}Pn \vdash \mathtt{andor}(0,0)\{P1, \ldots, Pn\}$

andor(*i*,*j*) (*i*$< n$, *j*> 0) -Elimination:

1. $\mathtt{andor}(i,j)\{P1, \ldots, Pn\}, P1, \ldots, Pj \vdash \tilde{}Pn$, for $0 \leq i \leq j < n$, $0 < j < n$,
2. $\mathtt{andor}(i,j)\{P_1, \ldots, P_n\}\ \tilde{}P_1, \ldots, \tilde{}P_{n-i} \vdash P_n$, for $0 < i < n$, $i \leq j \leq n$

Implication Elimination (a): $\mathtt{A}, \mathtt{A} => \mathtt{B} \vdash \mathtt{B}$.

Universal Elimination (a):

$A(a1, \ldots, an)$,
$\mathtt{all}(x1, \ldots, xn)(A(x1, \ldots, xn) => B(x1, \ldots, xn))$
$\vdash B(a1, \ldots, an)$