Visually Interacting with a Knowledge Base

Using Frames, Logic, and Propositional Graphs
With Extended Background Material

Daniel R. Schlegel and Stuart C. Shapiro

Department of Computer Science and Engineering
University at Buffalo, The State University of New York
Buffalo, New York, USA
{drschleg,shapiro}@buffalo.edu

Outline

1. Introduction
2. The Logic View
3. The Frame View
4. The Propositional Graph View
5. Retrieving Information
6. Collapsing the Graph
7. Geographic KB Example
8. Conclusions

SNePS 3

SNePS 3 is the latest member of the SNePS Family of KRR systems.

It is still being implemented.

The SNePS 3 KB can be thought of as simultaneously being:

- Logic based,
- Frame based, and
- Graph based.

We have created a user interface which uses all three:

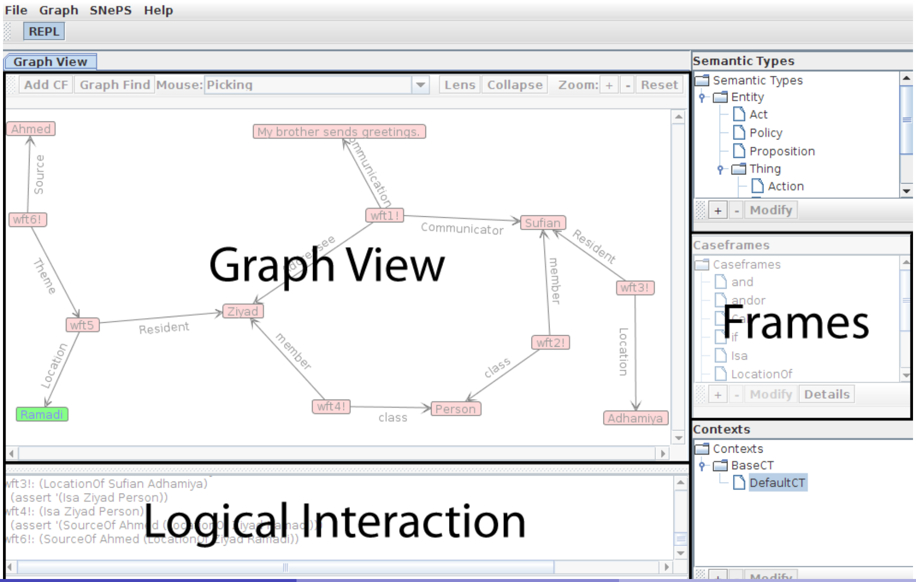
- Assertions and queries of a KB are handled using logic or frames.
- Visualization and inspection is done using propositional graphs.

Styles of Inference

Each view supports a style of inference:

- Logic-based view
  - Natural Deduction inference
- Frame-based view
  - Slot-based inference
- Graph-based view
  - Path-based inference

SNePS 3 GUI


File Graph SNePS Help
REPL
Graph View
Add CF Graph Find Mouse: Picking Lens Collapse Zoom: + - Reset
Ahmed
My brother sends greetings.
Source
wft6!
wft1!
Communicator
Sufian
Resident
Theme
Graph View
member
wft3!
wft5
Resident
Ziyad
wft2!
Location
Location
member
class
Ramadi
wft4!
class
Person
Adhamiya
wft3!: (LocationOf Sufian Adhamiya)
(assert '(Isa Ziyad Person))
wft4!: (Isa Ziyad Person)
wft6!: (SourceOf Ahmed (LocationOf Ziyad Ramadi))
Logical Interaction
Semantic Types
Semantic Types
Entity
Act
Policy
Proposition
Thing
Action
Modify
Caseframes
Caseframes
and
andor
Frames
if
Isa
LocationOf
Modify Details
Contexts
Contexts
BaseCT
DefaultCT
Modify

KB as set of Logical Expressions

The SNePS 3 KB is a set of logical expressions:

- Atomic terms
  - Individual constants denoting entities in domain including some relations
- Arbitrary and indefinite terms [Shapiro, KR2004]
- Functional terms including
  - terms denoting atomic propositions
  - terms denoting non-atomic propositions

Use CLIF syntax.
Every logical expression is a term.
Allows propositions about propositions without leaving First-Order.
Internal name of functional terms: `wft`*i* `[!]`
for “well-formed term”.

KB as set of Logical Expressions

The SNePS 3 KB is a set of logical expressions:

- Atomic terms
  - Individual constants denoting entities in domain including some relations
- Arbitrary and indefinite terms [Shapiro, KR2004]
- Functional terms
  including
  - terms denoting atomic propositions
  - terms denoting non-atomic propositions

Use CLIF syntax.
Every logical expression is a term.
Allows propositions about propositions without leaving First-Order.
Internal name of functional terms: `wft`*i* `[!]`
for “well-formed term”.

KB as set of Logical Expressions

The SNePS 3 KB is a set of logical expressions:

- Atomic terms
    - Individual constants denoting entities in domain including some relations
- Arbitrary and indefinite terms [Shapiro, KR2004]
- Functional terms
  including
    - terms denoting atomic propositions
    - terms denoting non-atomic propositions

Use CLIF syntax.
Every logical expression is a term.
Allows propositions about propositions without leaving First-Order.
Internal name of functional terms: `wft`*i* `[!]`
for "well-formed term".

KB as set of Logical Expressions

The SNePS 3 KB is a set of logical expressions:

- Atomic terms
  - Individual constants denoting entities in domain including some relations
- Arbitrary and indefinite terms [Shapiro, KR2004]
- Functional terms
  including
  - terms denoting atomic propositions
  - terms denoting non-atomic propositions

Use CLIF syntax.

Every logical expression is a term.

Allows propositions about propositions without leaving First-Order.

Internal name of functional terms: `wft`*i* `[!]`

for “well-formed term”.

KB as set of Logical Expressions

The SNePS 3 KB is a set of logical expressions:

- Atomic terms
  - Individual constants denoting entities in domain including some relations
- Arbitrary and indefinite terms [Shapiro, KR2004]
- Functional terms
  including
  - terms denoting atomic propositions
  - terms denoting non-atomic propositions

Use CLIF syntax.
Every logical expression is a term.
Allows propositions about propositions without leaving First-Order.
Internal name of functional terms: `wft`*i* `[!]`
for “well-formed term”.

KB as set of Logical Expressions

The SNePS 3 KB is a set of logical expressions:

- Atomic terms
  - Individual constants denoting entities in domain including some relations
- Arbitrary and indefinite terms [Shapiro, KR2004]
- Functional terms
  including
  - terms denoting atomic propositions
  - terms denoting non-atomic propositions

Use CLIF syntax.
Every logical expression is a term.
Allows propositions about propositions without leaving First-Order.
Internal name of functional terms: `wft`*i* `[!]`
for “well-formed term”.

Example Input

```
: (assert   '(Call Sufian Ziyad        "My brother sends greetings."))
wft1!: (Call Sufian Ziyad |My brother sends greetings.|)
: (assert '(Isa Sufian Person))
wft2!: (Isa Sufian Person)
: (assert '(LocationOf Sufian Adhamiya))
wft3!: (LocationOf Sufian Adhamiya)
: (assert '(Isa Ziyad Person))
wft4!: (Isa Ziyad Person)
: (assert '(SourceOf Ahmed (LocationOf Ziyad Ramadi)))
wft6!: (SourceOf Ahmed (LocationOf Ziyad Ramadi))
```

“Sufian, a person in Adhamiya, called Ziyad, a person who, according to Ahmed, is in Ramadi, saying ‘My brother sends greetings.’ ”

Note: `wft6` gives meta-information

Non-atomic Propositions

- `(not` p`)`
- `(thnot` p`)`
- `(and` $p_1, \ldots, p_n$`)`
- `(or` $p_1, \ldots, p_n$`)`
- `(nand` $p_1, \ldots, p_n$`)`
- `(nor` $p_1, \ldots, p_n$`)`
- `(xor` $p_1, \ldots, p_n$`)`
- `(iff` $p_1, \ldots, p_n$`)`
- `(thnor` $p_1, \ldots, p_n$`)`
- `(andor (`i j`)` $p_1, \ldots, p_n$`))`
- `(thresh (`i j`)` $p_1, \ldots, p_n$`)`
- `(if (setof` $p_1, \ldots, p_n$`) (setof` $q_1, \ldots, q_m$`))`
- `(v=> (setof` $p_1, \ldots, p_n$`) (setof` $q_1, \ldots, q_m$`))`
- `(`i`=> (setof` $p_1, \ldots, p_n$`) (setof` $q_1, \ldots, q_m$`))` [Shapiro, KR2010]

Natural Deduction Inference

- Forward-chaining and Backward-chaining
- Natural Deduction inference
- implemented
- but currently only for propositional fragment.

Example Forward Natural Deduction Inference

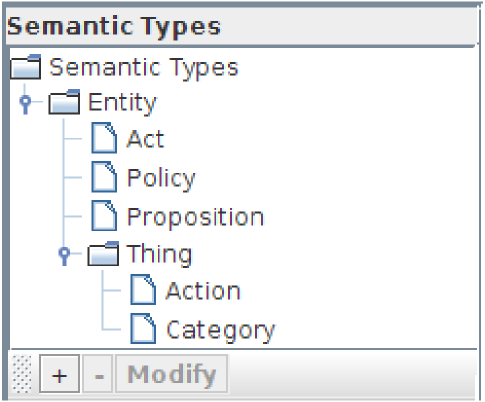
```
: (assert '(xor (Isa Pat Man) (Isa Pat Woman) (Isa Pat Robot)))
wft4!: (xor (Isa Pat Woman) (Isa Pat Man) (Isa Pat Robot))

: (assert! '(Isa Pat Woman))
Since wft4!: (xor (Isa Pat Woman) (Isa Pat Man) (Isa Pat Robot))
 and wft2!: (Isa Pat Woman)
I infer wft5!: (not (Isa Pat Man)) by Forward chaining.

Since wft4!: (xor (Isa Pat Woman) (Isa Pat Man) (Isa Pat Robot))
 and wft2!: (Isa Pat Woman)
I infer wft6!: (not (Isa Pat Robot)) by Forward chaining.

wft6!: (not (Isa Pat Robot))
wft5!: (not (Isa Pat Man))
wft2!: (Isa Pat Woman)
```

Example Natural Deduction Theorem Proving

```
: (ask '(if (if (and A B) C) (if A (if B C))))
Let me assume that wft2?: (if (and B A) C)
Let me assume that A
Let me assume that B
Since A
 and B
I infer wft1?: (and B A) by And Introduction.

Since wft2?: (if (and B A) C)
 and wft1?: (and B A)
I infer C by Implication Elimination.

Since C can be derived after assuming B
I infer wft3?: (if B C) by Implication Introduction.

Since wft3?: (if B C) can be derived after assuming A
I infer wft4?: (if A (if B C)) by Implication Introduction.

Since wft4?: (if A (if B C)) can be derived
                      after assuming wft2?: (if (and B A) C)
I infer wft5!: (if (if (and B A) C) (if A (if B C)))
        by Implication Introduction.

wft5!: (if (if (and B A) C) (if A (if B C)))
```

Sorted Logic

- Every term has a sort (semantic type).
- Sorts form a hierarchy.
- A sort may have multiple parents.
- User may introduce new sorts.
- Initial sort hierarchy (ontology):



The Sort of a Term

- May be specified at creation.
- May be inferred from use.
- Not represented in object language.
- Object language proposition, `(Isa term sort)`, is inferable from the sort hierarchy.

The Sort of a Term

- May be specified at creation.
- May be inferred from use.
- Not represented in object language.
- Object language proposition, `(Isa term sort)`, is inferable from the sort hierarchy.

The Sort of a Term

- May be specified at creation.
- May be inferred from use.
- Not represented in object language.
- Object language proposition, `(Isa term sort)`, is inferable from the sort hierarchy.

The Sort of a Term

- May be specified at creation.
- May be inferred from use.
- Not represented in object language.
- Object language proposition, `(Isa` *`term sort`*`)`, is inferable from the sort hierarchy.

Example Sort-based Inference

```
: (assert '(Isa Sufian Person))
wft1!: (Isa Sufian Person)

: (list-terms :types t)
<atom-Category> Person
<atom-Entity> Sufian
<categorization-Proposition> wft1!: (Isa Sufian Person)

: (ask '(Isa Person Category))
I infer wft2!: (Isa Person Category) by Sort-Based inference.
wft2!: (Isa Person Category)
```

Caseframes

- Based on “The Case for Case” [Fillmore, 1968] and The Berkeley FrameNet Project

  [Baker, Fillmore, & Lowe, 1998; Ruppenhofer *et al.*, 2010]
- Frame
  - schematic representation of a situation with a set of participants and conceptual roles.
- Eliminates syntactic differences.
- E.g.
  - Sufian called Ziyad.
  - Ziyad was called by Sufian.
  - a call from Sufian to Ziyad
- We will use “caseframe” for their “frame”
- and use “frame” for an instantiated caseframe.

Components of Caseframes

Definition

A caseframe has

- A name[a]
- A semantic type (sort)
  The type of the instances of the caseframe
- An ordered list of slots

[a] Temporary simplification for GUI.

Slots

Slots are defined globally
independently of the caseframes that use them.

Definition

A slot has

- A name
- A sort for its fillers
- Minimum and maximum number of fillers
- Adjustment rule: `reduce, expand, none`
- A path
- ...

Examples of Caseframes

Example

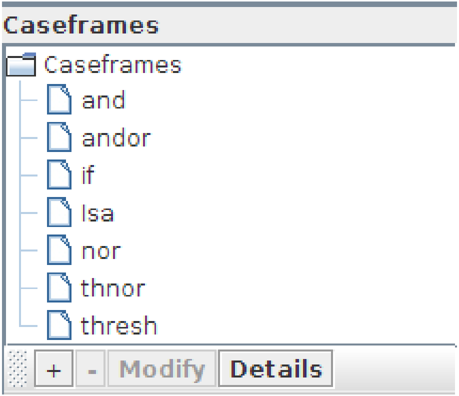
`Isa` is a caseframe of type `Proposition`
with slots `member` and `class`.

Example

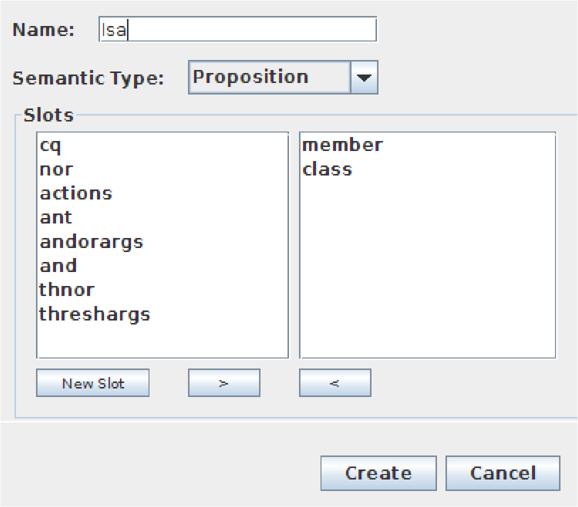
`Call` is a caseframe of type `Proposition`
with slots `Communicator`, `Addressee`, and `Communication`.

Caseframes available

The user interface maintains a list of the available caseframes for use.



Defining a caseframe in the GUI



Frames vs. Logical Terms

- A *frame* is an instance of a caseframe.
- The logical term `(F` $x_1, ..., x_n$`)`
  is represented by an instance of the caseframe named `F`
  whose slots, $s_1, ..., s_n$
  are filled by the representations of $x_1, ..., x_n$, respectively.

Frames vs. Logical Terms: Example

```
(assert '(Call Sufian Ziyad
               "My brother sends greetings"))
```

creates an instance of the `Call` caseframe

whose `Communicator` slot contains the filler `Sufian`,

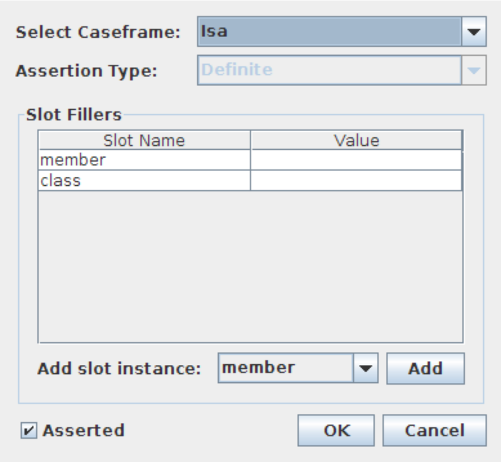
whose `Addressee` slot contains `Ziyad`,

and whose `Communication` slot

contains `"My brother sends greetings"`.

Assertions to the KB

Propositions (frames) can be added to the KB through a graphical interface much like filling in a database table.



Slot-Based Inference

A frame, F_1, logically entails another, F_2

if F_2's slots are filled with subsets or supersets of F_1's

according to the adjustment rules of the slots.

Example of Slot-based Inference

```
: (assert '(Isa (setof Fido Rover Lassie) (setof Dog Pet)))
wft1!: (Isa (setof Rover Lassie Fido) (setof Dog Pet))

: (ask '(Isa (setof Fido Rover) Dog))
Since wft1!: (Isa (setof Rover Lassie Fido) (setof Dog Pet))
I infer wft2!: (Isa (setof Rover Fido) Dog)
        by Slot-Based inference.

wft2!: (Isa (setof Rover Fido) Dog)
```

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

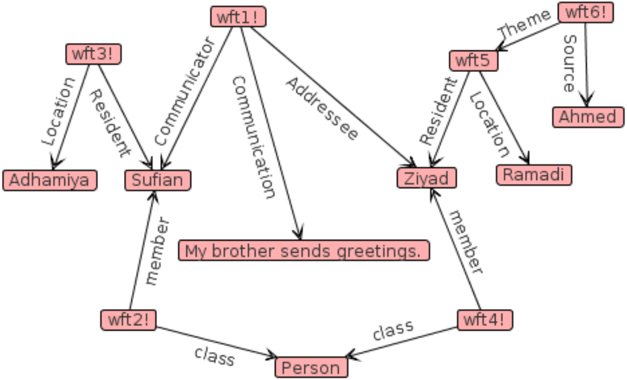
- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Propositional Graphs

A way of visualizing and traversing the frames.

- Directed Acyclic Graph (except for `some` and `every` arcs)
- Every term is a node.
  - Individual constants
  - Arbitrary and Indefinite terms
  - Functional terms (frames)
  - Proposition-denoting functional terms
- Node ID is
  - symbol
  - frame name (`wft`*i* [!])
- Edges drawn
  from the node corresponding to the frame,
  to the nodes corresponding to the slot fillers
- Edges labeled by slot names

Example Propositional Graph

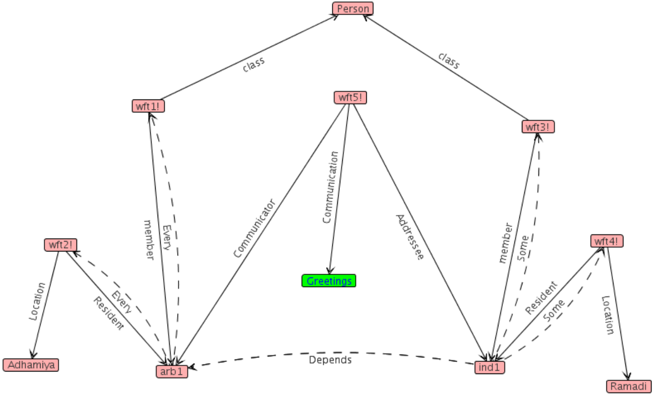


"Sufian, a person in Adhamiya, called Ziyad, a person who, according to Ahmed, is in Ramadi, saying 'My brother sends greetings.'"

Some GUI Facilities on Graph

- Can drag nodes.
- Can pan and zoom.
- Implemented in Jung

Arbitrary and Indefinite Terms




```
(Call (every x (Isa Person x) (LocationOf x Adhamya))
      (some y (x) (Isa Person y) (LocationOf y Ramadi))
      Greetings)
```

Path-Based Inference

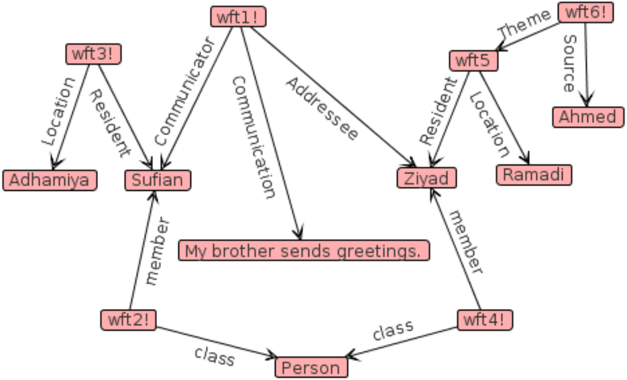
A propositional function node, p_2 is logically entailed by another, p_1

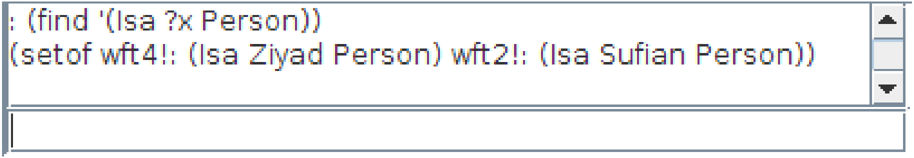
if an arc-node wire from p_2 is implied by

an appropriate path descending from p_1.

Currently broken in SNePS 3.

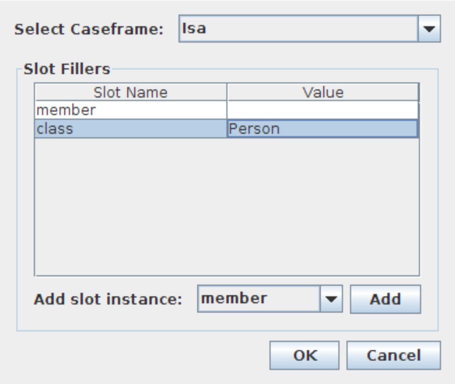
Logic-based `find`





Query By Example

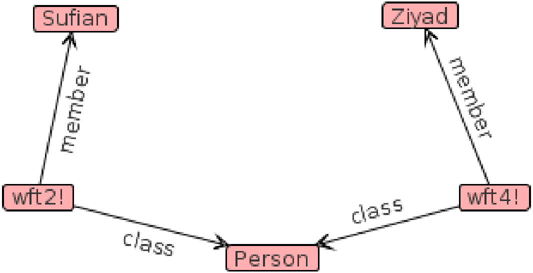
Using Frame view:



(Find and QBE don't currently do inference.)

Graphical Result of Query

Graphical view of result of `find` or QBE ("Filtered" Graph):



Query via the Graph (Inspection)

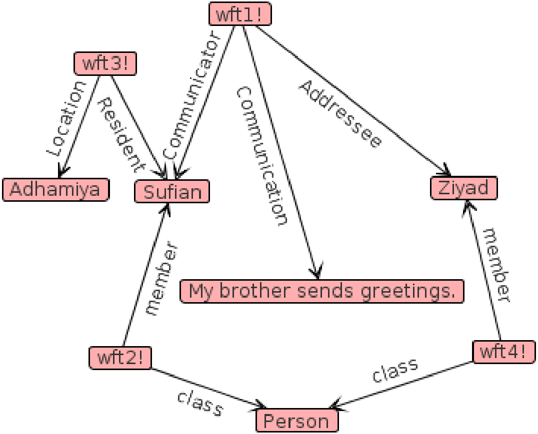
Right click on node.
Can:

- Hide node.
- Show/Hide frames it is a filler in.
- Show/Hide slots and fillers.

Show either all slots/fillers or none.

Example Result of Graph Query

Right click on `Sufian` node.
Choose to show frames it is a filler in.



Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Graph View as Visualization

- Visualized graph is for human comprehension.
- Visualized graph need not be isomorphic to implementation of KB.
- Usefulness of `wft` nodes:
  - Functional term with more than two arguments (slots).
  - Functional term with more than one filler in a slot.
  - Functional term shown as argument of another (filler in a slot).
- Can show a binary relation with no arc coming into it as a labeled arc (“collapsed arc”).

Visualizing a Collapsed Arc

- Slots in a frame are ordered.
- Order of slots = order of arguments of functional term.
- Draw collapsed arc from first argument to second argument.

- Name of caseframe = function symbol.
- Label collapsed arc with function symbol.

- Different style of arrow head
  so user knows it's a collapsed arc.

Visualizing a Collapsed Arc

- Slots in a frame are ordered.
- Order of slots = order of arguments of functional term.
- Draw collapsed arc from first argument to second argument.

- Name of caseframe = function symbol.
- Label collapsed arc with function symbol.

- Different style of arrow head
  so user knows it's a collapsed arc.

Visualizing a Collapsed Arc

- Slots in a frame are ordered.
- Order of slots = order of arguments of functional term.
- Draw collapsed arc from first argument to second argument.

- Name of caseframe = function symbol.
- Label collapsed arc with function symbol.

- Different style of arrow head
  so user knows it's a collapsed arc.

Visualizing a Collapsed Arc

- Slots in a frame are ordered.
- Order of slots = order of arguments of functional term.
- Draw collapsed arc from first argument to second argument.

- Name of caseframe = function symbol.
- Label collapsed arc with function symbol.

- Different style of arrow head
  so user knows it's a collapsed arc.

Visualizing a Collapsed Arc

- Slots in a frame are ordered.
- Order of slots = order of arguments of functional term.
- Draw collapsed arc from first argument to second argument.

- Name of caseframe = function symbol.
- Label collapsed arc with function symbol.

- Different style of arrow head
  so user knows it's a collapsed arc.

Visualizing a Collapsed Arc

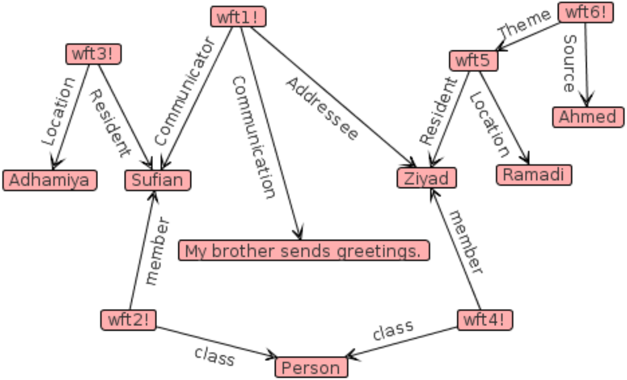
- Slots in a frame are ordered.
- Order of slots = order of arguments of functional term.
- Draw collapsed arc from first argument to second argument.

- Name of caseframe = function symbol.
- Label collapsed arc with function symbol.

- Different style of arrow head
  so user knows it's a collapsed arc.

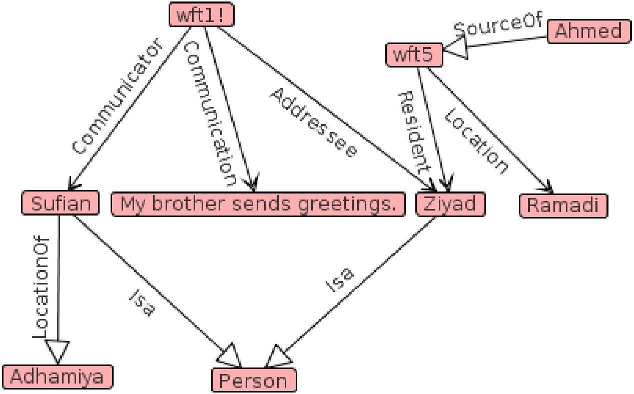
Example of Collapsed Graph: Before

The uncollapsed version of Suifian calling Ziyad example:



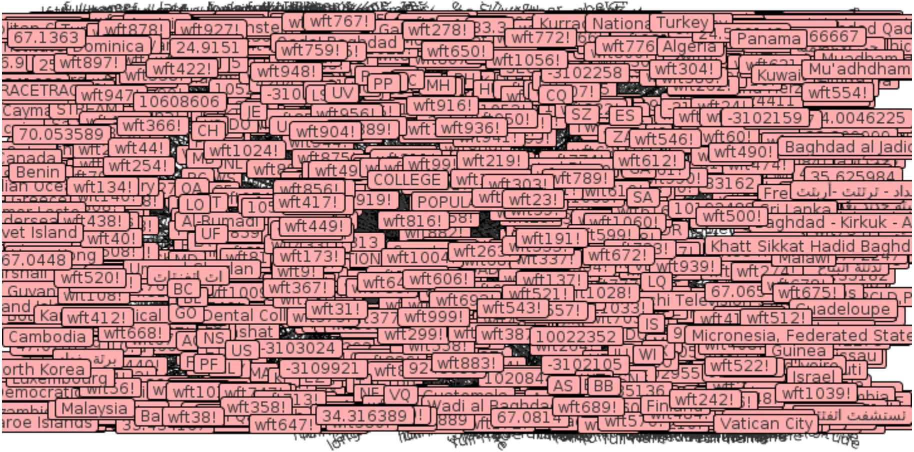
Example of Collapsed Graph: After

The collapsed version of Suifian calling Ziyad example:



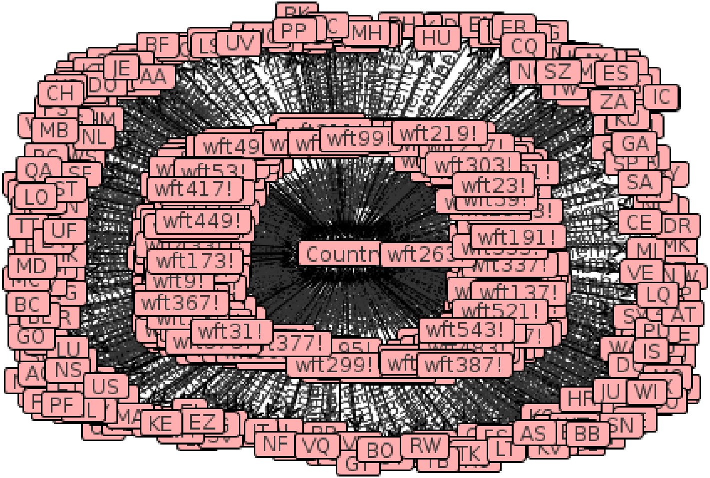
NGA GEOnet Names Server KB

2,075 terms representing place names and information about them.

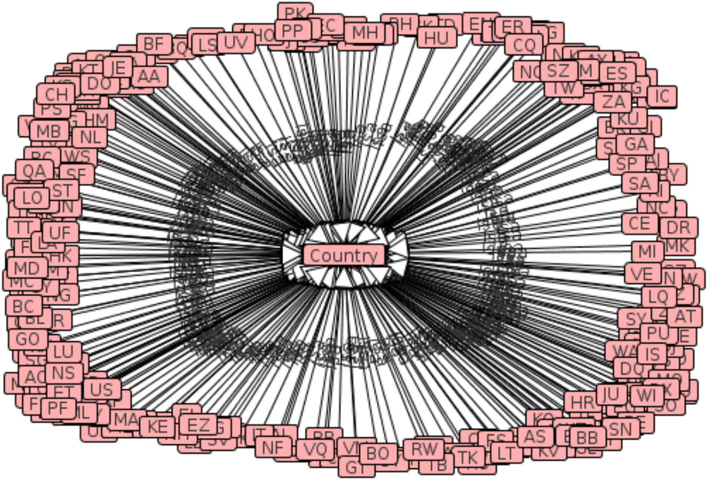


Filtered Graph

Filtered to show only instances of `(Isa ?x Country)`



Collapsed Filtered Graph



Evaluation of Graph Visualization

These techniques have been used successfully on graphs containing several thousand nodes.

The techniques should scale much further; the limitation may be the JUNG graphing system.

Conclusions

- Can view a Knowledge Base as
  - A set of logical expressions
  - A set of frames
  - A propositional graph
- Each view provides a style of inference.
- A GUI can supply all views,
- use whichever view is most appropriate for the purpose.

Acknowledgements

Multidisciplinary University Research Initiative (MURI) Grant
(Number W911NF-09-1-0392)
for “Unified Research on Network-based Hard/Soft Information Fusion”
Issued by the US Army Research Office (ARO)
under the program management of Dr. John Lavery.