CSE 663

Advanced Knowledge Representation

A Default Project of CVA

Final Report

Zhaomo Yang

1 Introduction

This project is a initial Context Vocabulary Acquisition (CVA) project. In this project, a short text passage will be represented by SNePSLOG, together with the some prior knowledge gained from some informants [1]. Then, the algorithm will be modified for the knowledge base built in SNePSLOG.

2 Project Plans

"1. Choose a passage containing a unfamiliar word.
2. Conduct an informal experiment with friends, asking them to read the passage and to figure out the meaning of the word. Make a record of how they figure out the meaning of the unfamiliar word.
3. Represent the passage and the prior knowledge from step 2 in SNePS.
4. Run the definition algorithm on the representation. In order to improve the performance of the definition algorithm, representation of the passage as well as the prior knowledge and the algorithm may be modified." [2]

3 The Text and the Unfamiliar Word

The text is given below, and the unfamiliar word is "comport", which is in bold in the passage.

> "There were, also, divers ladies in New York, Newport, and elsewhere, and celebrated for their palatial homes, their jewels, and their daughters, who were anxious to know how Bellew would **comport** himself under his disappointment. Some leaned to the idea that he would immediately blow his brains out; others opined that he would promptly set off on another of his exploring expeditions, and get himself torn to pieces by lions and tigers, or devoured by alligators; while others again feared greatly that, in a fit of pique, he would marry some "young person" unknown, and therefore, of course, utterly unworthy."[3]

4 Record of Interview

4.1 Definition of Unfamiliar word

My informants inferred the definition of “comport” in this context is “to do harm”.

4.2 Prior Knowledge used by informants

(1) “Kill” is a kind of “do harm”.
(2) If someone blows his brains out, then he kills himself.
(3) “Tear” is a kind of “kill”.
(4) “Devour” is a kind of “kill”.
(5) If someone sets off on an expedition, then he moves.
(6) For someone, if one of following three events happens: w1 does a1 to him; w2 does a2 to him; w3 does a3 to him. And a1 is an instance of v1, and a2 is an instance of v1, and a3 is an instance of v2, and v1 is a kind of "kill", v2 is a kind of "kill", then he is in danger.
(7) If someone moves and he is in danger now, then he does harm to himself.
(8) If someone does something which is unworthy, then he does harm to himself.
(9) If someone does a1 which is an instance of v1, and he also does a2 which is an instance of v2, and v1 is unknown, then v1 is a kind of v2.

5 Representations of the Text and the Prior Knowledge

5.1 Simplified Sentences

(1) Sentence 1

There were, also, divers ladies in New York, Newport, and elsewhere, and celebrated for their palatial homes, their jewels, and their daughters, who were anxious to know how Bellew would **comport** himself under his disappointment.

Simplified version:

Bellew **comports** himself.

(2) Sentence 2

Some leaned to the idea that he would immediately blow his brains out;

Simplified version:

Bellew blows his brains out.

(3) Sentence 3

others opined that he would promptly set off on another of his exploring expeditions, and get himself torn to pieces by lions and tigers, or devoured by alligators;

Simplified version:

Bellew sets off on another of his exploring expeditions, and get himself torn to

pieces by lions and tigers, or devoured by alligators.

(4) Sentence 4
while others again feared greatly that, in a fit of pique, he would marry some "young person" unknown, and therefore, of course, utterly unworthy.
Simplified version:
Bellew marries a young person who is unknown, which is unworthy.

5.2 Caseframes Used in Representation

(1) thing-called
define-frame thing-called (nil lex) "[lex] is the word of the name of the node which has an arc labeled lex pointing to [lex]"

(2) Named
define-frame Named (nil proper-name object) "[proper-name] is the name of [object]"

(3) act-wrt
define-frame act-wrt (nil action object) "it is an act whose action is [action] and whose object is [object]"

(4) the-action
define-frame the-action (nil action) “it is an act whose action is [action]”

(5) Does
define-frame Does (nil act agent) "[agent] does [act]"

(6) Isa-Part-Of
define-frame Isa-Part-Of (nil part whole) "[part] is a part of [whole]"

(7) AKO
define-frame AKO (nil subclass superclass) "[subclass] is a kind of [superclass]"
Currently the standard output of CVA contains the synonyms, superclass and properties of the unfamiliar word.

(8) Isa
define-frame Isa (nil member class) "[member] is a member of [class]"

(9) Is
define-frame Is (nil property object) "[object] is [property]"

(11) Status
define-frame Status (nil object status) "[object] is in [status] now"

(12) skolem1
define-frame skolem1 (func arg1 arg2 arg3) "[func] is an action whose arguments are [arg1], [arg2] and [arg3]"

(13) skolem2
define-frame skolem2 (func arg1 arg2 arg3) "[func] is an action whose arguments are [arg1], [arg2] and [arg3]"

(14) skolem3
define-frame skolem3 (func arg1 arg2) "[func] is an action whose arguments are [arg1] and [arg2]"

(15) skolem4
define-frame skolem4 (func arg1 arg2) "[func] is an action whose arguments are [arg1] and [arg2]"

The difference between caseframe “act-wrt” and “the-action” is that the former one is for transitive verb to fill the act filler of caseframe “Does”, while the latter one is for intransitive verb.

5.3 Representation of Prior Knowledge

Prior knowledge is what readers already know before reading the text, so it will be asserted in the system.

(1) “Kill” is a kind of “do harm”.
AKO(thing-called(kill), thing-called("do harm")).

(2) If someone blows his brains out, then he kills himself.
all(p, a, b)({Does(act-wrt(a, b), p), Isa(a, thing-called("blow out")), Isa(b, thing-called(brain)), Isa-Part-Of(b, p)} &=> {Does(act-wrt(skolem1(p, a, b), p), p), Isa(skolem1(p, a, b), thing-called(kill))}).

(3) “Tear” is a kind of “kill”.
AKO(thing-called(tear), thing-called("kill")).

(4) “Devour” is a kind of “kill”.
AKO(thing-called(devour), thing-called("kill")).

(5) If someone sets off on an expedition, then he moves.
all(p, a)({Does(the-action(a), p), Isa(a, thing-called("set off expedition"))} &=> {Does(the-action(skolem4(p, a)), p), Isa(skolem4(p, a), thing-called(move))}).

(6) For someone, if one of following three events happens: w1 does a1 to him; w2

does a2 to him; w3 does a3 to him. And a1 is an instance of v1, and a2 is an instance of v1, and a3 is an instance of v2, and v1 is a kind of "kill", v2 is a kind of "kill", then he is in danger.
all(w1, w2, w3, a1, a2, a3, p, v1, v2, l)({andor(1, 1){Does(act-wrt(a1, p), w1), Does(act-wrt(a2, p), w2), Does(act-wrt(a3, p), w3)}, Isa(a1, v1), Isa(a2, v1), Isa(a3, v2), AKO(v1, thing-called(kill)), AKO(v2, thing-called(kill))} &=> Status(p, thing-called("in danger"))).

(7) If someone moves and he is in danger now, then he does harm to himself.
all(p, a)({Does(the-action(a), p), Isa(a, thing-called(move)), Status(p, thing-called("in danger"))} &=> {Does(act-wrt(skolem3(p, a), p), p), Isa(skolem3(p, a), thing-called("do harm"))}).

(8) If someone does something which is unworthy, then he does harm to himself.
all(p, a, o)({Does(act-wrt(a, o), p), Is(thing-called(unworthy), Does(act-wrt(a, o), p))} &=> {Does(act-wrt(skolem2(p, a, o), p), p), Isa(skolem2(p, a, o), thing-called("do harm"))}).

(9) If someone does a1 which is an instance of v1, and he also does a2 which is an instance of v2, and v1 is unknown, then v1 is a kind of v2.
all(p, a1, a2, v1, v2)({Does(act-wrt(a1, p), p), Does(act-wrt(a2, p), p), Isa(a1, v1), Isa(a2, v2), Is(thing-called(unknown), v1)} &=> {AKO(v1, v2)}).

5.4 Representation of the Simplified Sentences

In order to simulate the way a reader reads a text, the text will be added to the system and forward inference will be fired.

(1) Bellew **comports** himself.
Named(Bellew, bellew)!
Does(act-wrt(a1, bellew), bellew)!
Isa(a1, thing-called(comport))!
Is(thing-called(unknown), thing-called(comport))!

(2) Bellew blows his brains out.
Isa(b, thing-called(brain))!
Isa-Part-Of(b, bellew)!
Does(act-wrt(a2, b), bellew)!
Isa(a2, thing-called("blow out"))!

(3) Bellew sets off on another of his exploring expeditions, and get himself torn to pieces by lions and tigers, or devoured by alligators
Does(the-action(a3), bellew)!

Isa(a3, thing-called("set off expedition"))!
andor(1, 1){Does(act-wrt(a4, bellew), tiger), Does(act-wrt(a5, bellew), lion),
Does(act-wrt(a6, bellew), aligator)}!
Isa(a4, thing-called(tear))!
Isa(a5, thing-called(tear))!
Isa(a6, thing-called(devour))!
Isa(tiger, thing-called(tiger))!
Isa(lion, thing-called(lion))!
Isa(aligator, thing-called(aligator))!

(4) Bellew marries a young person who is unknown, which is unworthy
Does(act-wrt(a7, p1), bellew)!
Isa(a7, thing-called(marry))!
Isa(p1, thing-called(person))!
Is(thing-called(young), p1)!
Is(thing-called(unknown), p1)!
Is(thing-called(unworthy), Does(act-wrt(a7, p1), bellew))!

6 Modified Verb Algorithm

6.1 Latest Verb Algorithms

Now the latest verb algorithm is written by Chris Becker [5]. There are two phases of processing in the algorithm: data collection and data processing. In the end, the algorithm will return a list of features of the unknown verb [6].

6.2 Why I Need to Modify the Algorithm

6.2.1 Knowledge base built via SNePSLOG

In this project, the knowledge base is built via SNePSLOG. Since formers CVA researchers built their knowledge bases via SNePSUL and therefore the algorithm is designed for that kind of knowledge base. So before using the verb algorithm, I need to modify it.

6.2.2 Different Way to Represent an Action Event

In this project, I represent an action event in a new and more reasonable way based on the suggestions of Prof. Shapiro and Prof. Rapaport. For example, if we want to represent the sentence that “Tom plays piano”.
Former researchers will represent it in this way:
Named(Tom, tom).
Does(act-wrt(thing-called(play), piano), tom).
Isa(piano, thing-called(Piano)).

However, I will treat the action as an instance of the concept of the verb:
Named(Tom, tom).
Does(act-wrt(play, piano), tom).
Isa(piano, thing-called(Piano)).
Isa(play, thing-called(play)).

Therefore, the current algorithm cannot retrieve all the useful information along the paths which are typical in former representation.

6.3 Implementation

6.3.1 Modify the algorithm for SNePSLOG

First, I put the modified algorithm in SNePSLOG package.
Second, all the with-SNePSUL reader macro, which are used to call SNePSUL commands in Lisp should be get rid of [4].
There are three usages in the file DataCollection_modified.cl

First:
Replace following codes

```
#3! ((find ~(first (rest (assoc rel (first (rest (assoc arg *Find-List*)))))) ~verb)
```

with

```
(eval `(sneps:find (first (rest (assoc rel (first (rest (assoc arg *Find-List*)))))) (^ verb)))
```

Second:
Replace following codes

```
#3! ((find ~(first (rest (assoc misc *Find-List*))) ~verb))
```

With

```
(eval `(sneps:find (first (rest (assoc misc *Find-List*))) (^ verb)))
```

Thrid:
Replace following codes

```
#3! ((find ~(first (rest (assoc arg *ArgsToFind*))) ~verb))
```

With

```
(eval `(sneps:find (first (rest (assoc arg *ArgsToFind*))) (^ verb)))
```

6.3.2 Modified the Path defined in the Algorithm

In the algorithm, a bunch of paths are defined to find useful information. For example, if we want to retrieve the node “agent” then we use the find function in sneps package with path named “agent” and node verb which is the unknown work in the passage. Then the find function will return the agent of the unknown verb. Since I applied a new way of representation, different paths should be designed for retrieving the same

argument.

Old path for retrieve the filler of arc “act”: (action lex)
New Path: (action member- class lex))

Old path for retrieve the filler of arc “agent”: (agent- act action lex)
New Path: (agent- act action member- class lex)

Old path for retrieve the filler of arc “object”: (object- action lex)
New Path: (object- action member- class lex)

Old path for retrieve the filler of arc “indobj”: (indobj- action lex))
New Path: (indobj- action member- class lex)

Old path for retrieve the filler of arc “from”: (from- action lex))
New Path: (from- action member- class lex)

Old path for retrieve the filler of arc “to”: (to- action lex)
New Path: (to- action member- class lex)

Old path for retrieve the filler of arc “instrument”: (instrument- act action lex)
New Path: (instrument- act action member- class lex)

Old path for retrieve the filler of arc “with”: (with- action lex))
New Path: (with- action member- class lex)

Old path for retrieve the filler of arc “in”: (in- action lex)
'New Path: (in- action member- class lex)

7 Sample Run

: demo "term_project_demo.txt"

File /home/unmdue/zhaomoya/term_project_demo.txt is now the source of input.

CPU time : 0.01

: ;
==========
; FILENAME: zhaomo-comport-demo.txt
; DATE: March 2
; PROGRAMMER: Zhaomo Yang

CPU time : 0.00

: ;; this template version:
;; http://www.cse.buffalo.edu/~rapaport/CVA/snepslog-template-2006114.demo

CPU time : 0.00

: ; Lines beginning with a semi-colon are comments.
; Lines beginning with "^" are Lisp commands.
; Lines beginning with "%" are SNePSUL commands.
; All other lines are SNePSLOG commands.
;
; To use this file: run SNePSLOG; at the SNePSLOG prompt (:), type:
;
;   demo "term_project_demo.txt" av
;
; Make sure all necessary files are in the current working directory
; or else use full path names.
;
==========

CPU time : 0.00

: ; Set SNePSLOG mode = 3
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 : 0.00

:

CPU time : 0.00

: ; Turn off inference tracing; this is optional.
; If tracing is desired, enter "trace" instead of "untrace":
untrace inference
Untracing inference.

CPU time : 0.00

:

CPU time : 0.00

: ; Load the appropriate definition algorithm:
^(cl:load "~/defun_verb_chris_modified.cl")
; Loading /home/unmdue/zhaomoya/defun_verb_chris_modified.cl
t

CPU time : 0.00

: ^(cl:load "~/DataProcessing_modified.cl")
; Loading /home/unmdue/zhaomoya/DataProcessing_modified.cl
t

CPU time : 0.01

: ^(cl:load "~/DataCollection_modified.cl")
; Loading /home/unmdue/zhaomoya/DataCollection_modified.cl
t

CPU time : 0.00

:

CPU time : 0.00

: ; Clear the SNePS network:
clearkb
Knowledge Base Cleared

CPU time : 0.00

:

CPU time : 0.00

: ; OPTIONAL:
; UNCOMMENT THE FOLLOWING CODE TO TURN FULL FORWARD INFERENCING ON:
;
^(cl:load "/projects/rapaport/CVA/STN2/ff.cl")
; Loading /projects/rapaport/CVA/STN2/ff.cl
Warning: broadcast-one-report, :operator was defined in /projects/snwiz/Install/Sneps-2.7.2/snip/fns/nrn-reports.lisp
and is now being defined in /projects/rapaport/CVA/STN2/ff.cl

```
t


  CPU time : 0.01

:


  CPU time : 0.00

:


  CPU time : 0.00

: ^(in-package snip)
#<The snip package>


  CPU time : 0.00

:


  CPU time : 0.00

: ^(defun broadcast-one-report (rep)
             (let (anysent)
                (do.chset (ch *OUTGOING-CHANNELS* anysent)
                          (when (isopen.ch ch)
                               (setq anysent (or (try-to-send-report
rep ch) anysent)))))
             nil)
snepslog::broadcast-one-report


  CPU time : 0.00

:


  CPU time : 0.00

: ^(in-package snepslog)
```

#<The snepslog package>

CPU time : 0.00

:

CPU time : 0.00

: ; Load all pre-defined relations:
^(sneps:intext "/projects/rapaport/CVA/verbalgorithm3.1/rels")
Loading file /projects/rapaport/CVA/verbalgorithm3.1/rels.
act is already defined.
action is already defined.
object1 is already defined.
object2 is already defined.
effect is already defined.
nil

CPU time : 0.02

:

CPU time : 0.00

: ; Load all pre-defined path definitions:
^(sneps:intext "/projects/rapaport/CVA/verbalgorithm3.1/paths")
Loading file /projects/rapaport/CVA/verbalgorithm3.1/paths.
before implied by the path (compose before (kstar (compose after- ! before)))
before- implied by the path (compose (kstar (compose before- ! after)) before-)
after implied by the path (compose after (kstar (compose before- ! after)))
after- implied by the path (compose (kstar (compose after- ! before)) after-)
sub1 implied by the path (compose object1- superclass- ! subclass superclass- ! subclass)
sub1- implied by the path (compose subclass- ! superclass subclass- ! superclass object1)
super1 implied by the path (compose superclass subclass- ! superclass object1- ! object2)
super1- implied by the path (compose object2- ! object1 superclass- ! subclass superclass-)
superclass implied by the path (or superclass super1)

superclass- implied by the path (or superclass- super1-)
nil

CPU time : 0.00

:

CPU time : 0.00

:

CPU time : 0.00

: ; define frames here:
; ====================
; (put annotated SNePSLOG code of your defined frames here;
; be sure to include both syntax and semantics)
; (also: be sure to define frames for any paths that you
; will need below!)
; (1) thing-called
define-frame thing-called (nil lex) "[lex] is the word of the name of the node"
thing-called(x1) will be represented by {<lex, x1>}

CPU time : 0.00

: ; (2) Named
define-frame Named (nil proper-name object) "[proper-name] is the name of [object]"
Named(x1, x2) will be represented by {<proper-name, x1>, <object, x2>}

CPU time : 0.00

: ; (3) act-wrt
define-frame act-wrt (nil action object) "it is an act whose action is [action] and whose object is [object]"
act-wrt(x1, x2) will be represented by {<action, x1>, <object, x2>}

CPU time : 0.00

: ; (4) the-action
define-frame the-action (nil action)
the-action(x1) will be represented by {<action, x1>}

CPU time : 0.00

: ; (5) Does
define-frame Does (nil act agent) "[agent] does [act]"
Does(x1, x2) will be represented by {<act, x1>, <agent, x2>}

CPU time : 0.00

: ; (6) Isa-Part-Of
define-frame Isa-Part-Of (nil part whole) "[part] is a part of [whole]"
Isa-Part-Of(x1, x2) will be represented by {<part, x1>, <whole, x2>}

CPU time : 0.00

: ; (7) AKO
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

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

CPU time : 0.00

: ; (9) Is
define-frame Is (nil property object) "[object] is [property]"
Is(x1, x2) will be represented by {<property, x1>, <object, x2>}

CPU time : 0.00

: ; (11) Status
define-frame Status (nil object status) "[object] is in [status] now"

Status(x1, x2) will be represented by {<object, x1>, <status, x2>}

CPU time : 0.00

: ; (12) skolem1
define-frame skolem1 (func arg1 arg2 arg3) "[func] is an action whose arguments are [arg1], [arg2] and [arg3]"
skolem1(x1, x2, x3) will be represented by {<func, skolem1>, <arg1, x1>, <arg2, x2>, <arg3, x3>}

CPU time : 0.00

: ; (13) skolem2
define-frame skolem2 (func arg1 arg2 arg3) "[func] is an action whose arguments are [arg1], [arg2] and [arg3]"
skolem2(x1, x2, x3) will be represented by {<func, skolem2>, <arg1, x1>, <arg2, x2>, <arg3, x3>}

CPU time : 0.00

: ; (14) skolem3
define-frame skolem3 (func arg1 arg2) "[func] is an action whose arguments are [arg1] and [arg2]"
skolem3(x1, x2) will be represented by {<func, skolem3>, <arg1, x1>, <arg2, x2>}

CPU time : 0.00

: ; (15) skolem4
define-frame skolem4 (func arg1 arg2) "[func] is an action whose arguments are [arg1] and [arg2]"
skolem4(x1, x2) will be represented by {<func, skolem4>, <arg1, x1>, <arg2, x2>}

CPU time : 0.00

:

CPU time : 0.00

: ; define paths here:
; ====================
; (put annotated SNePSLOG code for your paths here;
; be sure to include both syntax and semantics;
; consult "/projects/rapaport/CVA/mkb3.CVA/paths/snepslog-paths"
; for the proper syntax and some suggested paths;
; be sure to define frames above for any paths that you need here!)
;;; (1) If a has b as a superclass, and b has c as a superclass, then a has c as a superclass.
define-path superclass (or superclass
(compose ! superclass
(kstar (compose subclass- ! superclass))))
superclass implied by the path (or superclass (compose ! superclass (kstar (compose subclass- ! superclass))))
superclass- implied by the path (or superclass- (compose (kstar (compose superclass- ! subclass)) superclass- !))

CPU time : 0.00

:

CPU time : 0.00

: ;;; (2) If a has b as a subclass, and b has c as a subclass, then a has c as a subclass.
define-path subclass (or subclass
(compose ! subclass
(kstar (compose superclass- ! subclass))))
subclass implied by the path (or subclass (compose ! subclass (kstar (compose superclass- ! subclass))))
subclass- implied by the path (or subclass- (compose (kstar (compose subclass- ! superclass)) subclass- !))

CPU time : 0.00

:

CPU time : 0.00

: ; BACKGROUND KNOWLEDGE:
; ====================

; (1) Killing is a kind of doing harm.
AKO(thing-called(kill), thing-called("do harm")).

wff3!: AKO(thing-called(kill),thing-called(do harm))

CPU time : 0.00

:

CPU time : 0.00

: ; (2) If someone blows his brains out, then he kills himself.
all(p, a, b)({Does(act-wrt(a, b), p), Isa(a, thing-called("blow out")), Isa(b, thing-called(brain)), Isa-Part-Of(b, p)} &=> {Does(act-wrt(skolem1(p, a, b), p), p), Isa(skolem1(p, a, b), thing-called(kill))}).

wff6!:
all(b,a,p)({Isa-Part-Of(b,p),Isa(b,thing-called(brain)),Isa(a,thing-called(blow out)),Does(act-wrt(a,b),p)} &=> {Isa(skolem1(p,a,b),thing-called(kill)),Does(act-wrt(skolem1(p,a,b),p),p)})

CPU time : 0.00

:

CPU time : 0.00

: ; (3) Tearing is a kind of killing.
AKO(thing-called(tear), thing-called("kill")).

wff8!: AKO(thing-called(tear),thing-called(kill))

CPU time : 0.00

:

CPU time : 0.00

: ; (4) Devouring is a kind of killing.
AKO(thing-called(devour), thing-called("kill")).

wff10!: AKO(thing-called(devour),thing-called(kill))

CPU time : 0.00

:

CPU time : 0.00

: ; (5) If someone sets off on a expedition, then he moves.
all(p, a)({Does(the-action(a), p), Isa(a, thing-called("set off expedition"))} &=> {Does(the-action(skolem4(p, a)), p), Isa(skolem4(p, a), thing-called(move))}).

wff13!: all(a,p)({Isa(a,thing-called(set off expedition)),Does(the-action(a),p)} &=> {Isa(skolem4(p,a),thing-called(move)),Does(the-action(skolem4(p,a)),p)})

CPU time : 0.00

:

CPU time : 0.00

: ; (6) For someone, if one of following three events happens: w1 does a1 to him; w2 does a2 to him; w3 does a3 to him. And a1 is an instance of v1, and a2 is an instance of v1, and a3 is an instance of v2, and v1 is a kind of "kill", v2 is a kind of "kill", then he is in danger.
all(w1, w2, w3, a1, a2, a3, p, v1, v2, l)({andor(1, 1){Does(act-wrt(a1, p), w1), Does(act-wrt(a2, p), w2), Does(act-wrt(a3, p), w3)}, Isa(a1, v1), Isa(a2, v1), Isa(a3, v2), AKO(v1, thing-called(kill)), AKO(v2, thing-called(kill))} &=> Status(p, thing-called("in danger"))).

wff15!:
all(l,v2,v1,p,a3,a2,a1,w3,w2,w1)({AKO(v2,thing-called(kill)),AKO(v1,thing-called(kill)),Isa(a3,v2),Isa(a2,v1),Isa(a1,v1),andor(1,1){Does(act-wrt(a3,p),w3),Does(act-wrt(a2,p),w2),Does(act-wrt(a1,p),w1)}} &=> {Status(p,thing-called(in danger))})

CPU time : 0.00

:

CPU time : 0.00

: ; (7) If someone moves and he is in danger now, then he does harm to him self.
all(p, a)({Does(the-action(a), p), Isa(a, thing-called(move)), Status(p, thing-called("in danger"))} &=> {Does(act-wrt(skolem3(p, a), p), p), Isa(skolem3(p, a), thing-called("do harm"))}).

wff16!: all(a,p)({Status(p,thing-called(in danger)),Isa(a,thing-called(move)),Does(the-action(a),p)} &=> {Isa(skolem3(p,a),thing-called(do harm)),Does(act-wrt(skolem3(p,a),p),p)})

CPU time : 0.00

:

CPU time : 0.00

: ; (8) If someone does something which is unworthy, then he does harm to himself.
all(p, a, o)({Does(act-wrt(a, o), p), Is(thing-called(unworthy), Does(act-wrt(a, o), p))} &=> {Does(act-wrt(skolem2(p, a, o), p), p), Isa(skolem2(p, a, o), thing-called("do harm"))}).

wff18!:
all(o,a,p)({Is(thing-called(unworthy),Does(act-wrt(a,o),p)),Does(act-wrt(a,o),p)} &=> {Isa(skolem2(p,a,o),thing-called(do harm)),Does(act-wrt(skolem2(p,a,o),p),p)})

CPU time : 0.00

:

CPU time : 0.00

: ; (9) If someone does a1 which is an instance of v1, and he also does a2 which is an instance of v2, and v1 is unknown, then v1 is a kind of v2.
all(p, a1, a2, v1, v2)({Does(act-wrt(a1, p), p), Does(act-wrt(a2, p), p), Isa(a1, v1), Isa(a2, v2), Is(thing-called(unknown), v1)} &=> {AKO(v1, v2)}).

wff20!:
all(v2,v1,a2,a1,p)({Is(thing-called(unknown),v1),Isa(a2,v2),Isa(a1,v1),Does(act-wrt(a2,p),p),Does(act-wrt(a1,p),p)} &=> {AKO(v1,v2)})

CPU time : 0.00

:

CPU time : 0.00

:

CPU time : 0.00

: ; CASSIE READS THE PASSAGE:
; ==========
; (1) Bellew comports himself under his disappointment.
Named(Bellew, bellew)!

wff21!: Named(Bellew,bellew)

CPU time : 0.00

: Does(act-wrt(a1, bellew), bellew)!

wff25!: Does(the-action(a1),bellew)
wff23!: Does(act-wrt(a1,bellew),bellew)
wff10!: AKO(thing-called(devour),thing-called(kill))
wff8!: AKO(thing-called(tear),thing-called(kill))

CPU time : 0.10

: Isa(a1, thing-called(comport))!

wff37!: Isa(a1,thing-called(comport))

CPU time : 0.01

: Is(thing-called(unknown), thing-called(comport))!

wff38!: Is(thing-called(unknown),thing-called(comport))

CPU time : 0.01

:

CPU time : 0.00

: ; (2) Bellew blows his brains out.
Isa(b, thing-called(brain))!

wff39!: Isa(b,thing-called(brain))

CPU time : 0.01

: Isa-Part-Of(b, bellew)!

wff40!: Isa-Part-Of(b,bellew)

CPU time : 0.01

: Does(act-wrt(a2, b), bellew)!

wff44!: Does(the-action(a2),bellew)
wff42!: Does(act-wrt(a2,b),bellew)

CPU time : 0.01

: Isa(a2, thing-called("blow out"))!

wff52!: AKO(thing-called(comport),thing-called(kill))
wff51!: Does(the-action(skolem1(bellew,a2,b)),bellew)
wff49!: Isa(skolem1(bellew,a2,b),thing-called(kill))
wff48!: Does(act-wrt(skolem1(bellew,a2,b),bellew),bellew)
wff45!: Isa(a2,thing-called(blow out))

CPU time : 0.07

:

CPU time : 0.00

: ; (3) Bellew sets off on another of his exploring expeditions, and get himself torn to pieces by lions and tigers, or devoured by alligators
Does(the-action(a3), bellew)!

wff55!: Does(the-action(a3),bellew)

CPU time : 0.01

: Isa(a3, thing-called("set off expedition"))!

wff60!: Does(the-action(skolem4(bellew,a3)),bellew)
wff58!: Isa(skolem4(bellew,a3),thing-called(move))
wff56!: Isa(a3,thing-called(set off expedition))

CPU time : 0.03

: andor(1, 1){Does(act-wrt(a4, bellew), tiger), Does(act-wrt(a5, bellew), lion), Does(act-wrt(a6, bellew), aligator)}!

wff67!:
andor(1,1){Does(act-wrt(a6,bellew),aligator),Does(act-wrt(a5,bellew),lion),Does(act-wrt(a4,bellew),tiger)}

CPU time : 0.07

: Isa(a4, thing-called(tear))!

wff71!: Isa(a4,thing-called(tear))

CPU time : 0.01

: Isa(a5, thing-called(tear))!

wff72!: Isa(a5,thing-called(tear))

CPU time : 0.01

: Isa(a6, thing-called(devour))!

wff79!: Does(the-action(skolem3(bellew,skolem4(bellew,a3))),bellew)
wff77!: Isa(skolem3(bellew,skolem4(bellew,a3)),thing-called(do harm))
wff76!: Does(act-wrt(skolem3(bellew,skolem4(bellew,a3)),bellew),bellew)
wff73!: Isa(a6,thing-called(devour))
wff53!: AKO(thing-called(comport),thing-called(do harm))
wff28!: Status(bellew,thing-called(in danger))

CPU time : 0.08

: Isa(tiger, thing-called(tiger))!

wff81!: Isa(tiger,thing-called(tiger))

CPU time : 0.01

: Isa(lion, thing-called(lion))!

wff83!: Isa(lion,thing-called(lion))

CPU time : 0.02

: Isa(aligator, thing-called(aligator))!

wff85!: Isa(aligator,thing-called(aligator))

CPU time : 0.03

:

CPU time : 0.00

: ; (4) Bellew marries a young person who is unknown, which is unworthy
Does(act-wrt(a7, p1), bellew)!

wff89!: Does(the-action(a7),bellew)
wff87!: Does(act-wrt(a7,p1),bellew)

CPU time : 0.02

: Isa(a7, thing-called(marry))!

wff91!: Isa(a7,thing-called(marry))

CPU time : 0.01

: Isa(p1, thing-called(person))!

wff93!: Isa(p1,thing-called(person))

CPU time : 0.01

: Is(thing-called(young), p1)!

wff95!: Is(thing-called(young),p1)

CPU time : 0.01

```
: Is(thing-called(unknown), p1)!

   wff96!:    Is(thing-called(unknown),p1)

  CPU time : 0.01

: Is(thing-called(unworthy), Does(act-wrt(a7, p1), bellew))!

   wff103!:    Does(the-action(skolem2(bellew,a7,p1)),bellew)
   wff101!:    Isa(skolem2(bellew,a7,p1),thing-called(do harm))
   wff100!:    Does(act-wrt(skolem2(bellew,a7,p1),bellew),bellew)
   wff97!:    Is(thing-called(unworthy),Does(act-wrt(a7,p1),bellew))
   wff53!:    AKO(thing-called(comport),thing-called(do harm))

  CPU time : 0.06

:


  CPU time : 0.00

: ; Ask Cassie what "comport" means:
^(defineVerb 'comport)
  verb:
          lex:    comport;
          superclass:    do harm; kill;
          property:    unknown;
          transitivity: (transitive) nil


  CPU time : 0.04

:

End of /home/unmdue/zhaomoya/term_project_demo.txt demonstration.


  CPU time : 0.73

:
```

8 Acknowledgements

I appreciate the help and advice of Prof. Shapiro and Prof. Rapaport.

9 References

[1] Rapaport, William J.; & Kibby, Michael W. (2007), "*Contextual Vocabulary Acquisition as Computational Philosophy and as Philosophical Computation*", *Journal of Experimental and Theoretical Artificial Intelligence* 19(1) (March): 1-17.

[2] Rapaport, William J. *Default CVA project*, http://www.cse.buffalo.edu/~rapaport/717cva/cvaproject.html

[3] Becker, Chris (2004). *"Contextual Vocabulary Acquisition; Contextual Information in Verb Contexts: From Analysis to Algorithm"* http://www.cse.buffalo.edu/~rapaport/CVA/becker-verbs.pdf

[4] Shapiro, S. C. and The SNePS Implementation Group (2007). *SNePS 2.7 User's Manual*. **http://www.cse.buffalo.edu/sneps/Manuals/manual27.pdf**.

[5] Becker, Chris (2005). "*defun_verb.cl", "DataCollection.cl" and "DataProcessing.cl"* /projects/rapaport/CVA/verbalgorithm3.1/

[6] Becker, Chris (2005). "*Contextual Vocabulary Acquisition of verb"* http://www.cse.buffalo.edu/~rapaport/CVA/Verbs/becker-revisedfinalreport.pdf