Advanced Topics

A bluffer’s guide to Cyc-style KBs
Dumping sketch knowledge to files
Extending the knowledge base
Making queries interactively
A KQML API for connecting CogSketch to other software
OpenCyc Knowledge Base

- Cyc = World’s largest and most complete general knowledge base
  - Hundreds of thousands of terms
  - Millions of assertions
  - English strings corresponding to many concept terms

- OpenCyc = open-source subset of Cyc
  - Entire ontology
  - Structural facts

- Two ways to explore
  - Download OpenCyc from SourceForge
    - Will not have the QRG extensions to OpenCyc
  - Use the browsing capabilities built into CogSketch

OpenCyc.org
Collections and Genls

- Concepts and categories in OpenCyc are modeled as *collections*.
- Collections are related to each other through the *genls hierarchy*.
- You can have instances of collections.
- Collection names begin with capital letters.

**Diagram:**

- **Dog**
  - genls **CanineGenus**
  - genls **DomesticatedAnimal**
  - genls **Collie**

**Notes:**

- *Dog* is the collection of all dogs.
  - (genls **Collie** Dog) also
  - (genls **Beagle** Dog)

- Everything that is an instance of **Collie** is also an instance of **Dog** but not vice versa.

- **Collie** is the collection of all dogs of the breed **Collie**.
Individuals

• An *individual* is a single thing, not a collection
• Individuals do not have instances
• Use *isa* to relate an individual to a collection

Lassie1 is an instance of the collection Dog.

Lassie1 is also an individual.
Predicates and genlPreds

- **Predicates** are used to build *sentences*
- A sentence built with a predicate is either true or false
- Predicate names begin with lower-case letters
- **genlPreds** indicates a hierarchical relationship between predicates

(owns Timmy1 Lassie1)

True

(biologicalRelatives Timmy1 Lassie1)

False

Predicates can also relate Collections

(animalTypeMakesSoundType Dog BarkingSound)

(disjointWith Cat Dog)
Arity and Argument Types

• Every predicate has two central features:
  – *Arity*: How many arguments does it require?
  – *Argument types*: What types of arguments does it require?
    • arg1Isa: SocialBeing
    • arg2Isa: SomethingExisting

• Every sentence must be both *semantically* and *syntactically* well-formed

Predicate: **owns**
  arity: 2
  arg1Isa: SocialBeing
  arg2Isa: SomethingExisting

(owns Timmy1 Lassie1)

(owns Timmy1 Lassie1 Rover2)

(owns Timmy1 Dog)

OK!

**Syntactically** poorly-formed

**Semantically** poorly-formed
Microtheories

- The knowledge in OpenCyc is organized into *Microtheories*
  - Assertions **within** a microtheory must be mutually consistent
  - Assertions in **different** microtheories may be inconsistent

Can separate statements based on:
- Time, source, granularity, ...

Microtheories Example:

```
TimmyInWellMT
(objectFoundInLocation Timmy1 OldWell1)
(isa Lassie1 Dog)
...

Inconsistent but in different Microtheories

TimmyEatsDinnerMT
(objectFoundInLocation Timmy1 Home1)
(isa Lassie1 Dog)
...
```
Using Microtheories

• To make a new microtheory
  - (isa TimmyInWellMT Microtheory)

• To relate one microtheory to another
  - (genlMt TimmyInWellMT LassieMT)

  Every assertion that is true in LassieMT is also true in TimmyInWellMT

• To make a statement on a microtheory
  - (ist-Information LassieMT (isa Lassie1 Dog))

  The assertion (isa Lassie Dog) is true in the microtheory LassieMT
Exporting knowledge to files

Export sketch to file

Sketch to be Exported:
Shopping Cart Redux

In what format should the knowledge be exported?

- MELD (CYC)
- MELD (CYC)
- FIRE (reloadable into FIRE's KB)
- KIF (Knowledge Interface Format)
- sketch-facts.txt

Fact Filter:
none

Select States

Include detailed ink descriptions?

OK
MELD format files

- Similar to Cyc KE format

```plaintext
;;; constant: Case-3429195339.
;;; in Mt: BaseKB.
(isa Case-3429195339 Microtheory)
(isa Case-3429195339 COASpecificationMicrotheory)
(genlMt Case-3429195339 BaseKB)

;;; constant: BCase-3429195452.
;;; in Mt: BaseKB.
(isa BCase-3429195452 Microtheory)
(isa BCase-3429195452 COASpecificationMicrotheory)
(genlMt BCase-3429195452 Case-3429195339)

;;; Default Mt: Case-3429195339.
```
FIRE format

- Pure Lisp syntax
- Microtheory toggled by directives in file
  - cf. KB extension example

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Shopping Cart Redux

(genlMt BCase-3429195452 Case-3429195339)
(genlMt Case-3429195339 SKEAREasoningCollectorMt)

(ist-Information Case-3429195339
 (askConceptualForBinaryVisualRelation Case-3429195339 BCase-3429195452
  Object-145 Object-147 rcc8-EC PhysicalView-SubSketch
  LookingFromSide-SubSketch))
(ist-Information Case-3429195339
 (askConceptualForBinaryVisualRelation Case-3429195339 BCase-3429195452
  Object-145 Object-148 rcc8-EC PhysicalView-SubSketch
  LookingFromSide-SubSketch))
Extending the Knowledge Base

- OpenCyc has a lot of knowledge ... but it might not have everything you need
- You add knowledge using a .meld file
- Create using your favorite text editor.

Hint: Use an editor that matches parentheses, such as emacs!
Example: A Simple Flat-File

(in-microtheory TimmyInWellMT) ;; Tells file
;; loader what microtheory to use. All forms after
;; this command are facts for that microtheory.
(isa Lassie1 Dog)
(isa Timmy1 MaleChild)
(isa OldWell1 Well)
(owns Timmy1 Lassie1)
(objectInLocation Timmy1 OldWell1)
(isa LassieGetHelp RescuingSomeone)
(performedBy LassieGetHelp Lassie1)
(beneficiary LassieGetHelp Timmy1)
Adding a Collection

To add a collection you need at least three things:
1. A statement that it is a Collection
2. A genls statement
3. A comment describing the collection

(isa Firefly Collection)
(genls Firefly Insect)
(comment Firefly “the collection of all insects that having glowing posteriors”)

QRG
To add a relation you need at least four things:
1. A statement that it is a Relation
2. An arity statement
3. ArgIsa statements
4. A comment describing the relation

(isa aboveGrazingLine Relation)
(arity aboveGrazingLine 2)
(arg1Isa aboveGrazingLine NuSketchGlyph)
(arg2Isa aboveGrazingLine NuSketchGlyph)
(comment aboveGrazingLine “the figure object represented by the glyph in arg1 is above the grazing line created by the ground object represented by the glyph in arg2”)
Using Your New KB entries in CogSketch

• Your new collections
  – Can be used in conceptual labeling
  – Can be used to constrain arguments to relations

• Your new relations
  – Can show up as hypothesized visual/conceptual relationship questions, if you weave them into the \texttt{genlPreds} hierarchy correctly.
  – Can be used for your own reasoning, if you add Horn clause axioms involving them also
    • Via browser query window, or API calls
    • Documentation on doing this is in progress
Querying the KB

Bottom left of Knowledge Inspector page

- Can type in queries to the reasoner of a sketch
- If you've edited the sketch, click this.
- Invokes a KB browser
Example: Browsing

- Let’s look for other relationships involving rotation with the KB browser

Possible matches for "rotat":
- Rotataion-None (Collection)
- RotatedShape-180 (Collection)
- RotatedShape-45 (Collection)
- RotatedShape-90 (Collection)
- RotatedShape-None (Collection)
- Rotation-135 (Collection)
- Rotation-180 (Collection)
- Rotation-45 (Collection)
- Rotation-90 (Collection)
- Rotation-Clockwise135 (Collection)
- Rotation-Clockwise45 (Collection)
- Rotation-Clockwise90 (Collection)
- Rotation-CounterClockwise135 (Collection)
**rotationallyConnectedTo [type = Relation]:**

comment: A ConnectionPredicate (q.v.) and thus a specialization of connectedTo (q.v.). (rotationallyConnectedTo OBJ1 OBJ2) means that OBJ1 and OBJ2 are connected in such a way that rotational motion, and only rotational motion, can happen between them. The range of rotational motion possible might be full or partial. Non-rotational movement between two rotationally connected objects can occur only if the connection is broken, deformed, or disassembled. If OBJ1 and OBJ2 do rotate relative to one another, then this may be due to sliding of their surfaces, articulation of some joint part, or deformation of OBJ1 or OBJ2 (so long as that deformation only allows rotation between OBJ1 and OBJ2). Positive examples: Femurs are rotationally connected to hips, doors are rotationally connected to door frames, doorknobs are rotationally connected to doors, and propellers are rotationally connected to airplanes; in computer trackballs the ball is rotationally connected to the housing. Also a book cover is rotationally connected to its binding (but flapHingedTo is even more appropriate for describing such a connection because it is more specific). Negative examples: a planet orbiting a star (they are not connected; cf. MovingInACircle) and a toothpick stuck in a person's leg (although elastic deformation of flesh allows there to be rotational motion between toothpick and leg, it also may allow a small amount of translational motion to occur between them; in-Lodged is more appropriate for describing this case).

*isa:*

in UniversalVocabularyMt: ConnectionPredicate, IrreflexiveBinaryPredicate, SymmetricBinaryPredicate  
in TopicMt: Connections-Spatial-Topic

arity: 2
arg1isa: SolidTangibleThing  
arg2isa: SolidTangibleThing

genlPreds:

in BaseKB: rotationallyConnectedTo  
in UniversalVocabularyMt: connectedTo

specPreds:

in UniversalVocabularyMt: connectedByBeltTo, hingedTo, screwedIn
Making Queries

Shopping Cart Redux
Case-3429195339

State Shopping Cart Anatomy
BCase-3429195452

- **Layer Positional**
  - ObjectL-226
- **Layer Voronoi**
  - ObjectL-224
- **Layer Physical**
  - ObjectL-225

Handle
  - Object-154
Front leg
  - Object-153
rear leg
  - Object-152
Body
  - Object-151
Front axle
  - Object-150
rear axle
  - Object-149
rear wheel
  - Object-148
front wheel
  - Object-147

Query

Enter your query here:

(isa Object-147 SolidTangibleThing)

Context: EverythingPSC

Facts: all

- Allow microtheory inheritance? (env)
- Allow gens inferencing? (transitive)
- Allow other kinds of inference? (infer)

Query using fire:ask  Query using fire:query
Can get Answers

Query

(isa Object-147 SolidTangibleThing)
query-type = ask
context = EverythingPSC; facts = all; env; transitive; infer

Answers:

? A (isa Object-147 SolidTangibleThing)

Ask New Query
Can Drill Down for Reasons

(isa Object-147 SolidTangibleThing)

The above expression is true because of the following:

? A (isa Object-147 Wheel) [true]

It is true via:

(:implied-by
 (:implies
   (:implies
     (ist-Information EverythingPSC (isa Object-147 Wheel))
     (ist-Information EverythingPSC
      (isa Object-147 SolidTangibleThing)))
   :transitive-isa)

Direct Consequences: NONE

Legend:
=X = Retract Fact  ? = Show Justifications
What is an API? Why do I want one?

• **Application Programming Interface**
• Allows you to access CogSketch from code
• Socket-based, using KQML messages
• Documentation and sample client provided with CogSketch executable
What Can I do with the API?

- Manipulate Sketches
  - (list-open-sketches)
  - (get-active-sketch)
  - (set-active-sketch :sketch-id <sketch id>)
  - (save-sketch-to-file :sketch-id <sketch id>)
  - (close-sketch :sketch-id <sketch id>)
  - (open-sketch-from-file :filepath <full path to file (string)>)
  - (create-new-sketch)
  - (name-of-sketch :sketch-id <sketch id>)
  - (user-namestring-of-sketch :sketch-id <sketch id>)
What Can I do with the API?

- You can also manipulate subsketches, Layers and Glyphs
  - (list-bundles :sketch-id <sketch id>)
  - (list-layers :sketch-id <sketch id> :bundle-id <bundle id>)
  - (name-of-layer :sketch-id <sketch id> :layer-id <layer id>)
  - (kind-of-layer :sketch-id <sketch id> :layer-id <layer id>)
  - (list-glyphs :sketch-id <sketch id> :layer-id <layer id>)
  - (delete-glyph :sketch-id <sketch id> :glyph-id <glyph id>)

- These are just examples of some of the available commands
Visual/Conceptual Relationships

- People use conventions for depicting physical relationships in sketches
- You can tell CogSketch about your assumptions
Example: Shopping Cart

(GlyphFn Object-147 User-Drawn-Sketch-Layer-225)

human-readable namestring: front wheel
glyph represents: Object-147

- isa [6 facts]
  - ? A (isa Object-147 Entity)
  - ? A (isa Object-147 Wheel)

- spatiallyIntersects [4 facts]
  - ? A (spatiallyIntersects (GlyphFn Object-147 User-Drawn-Sketch-Layer-225) (GlyphFn Object-153 User-Drawn-Sketch-Layer-225))
  - ? A (spatiallyIntersects (GlyphFn Object-147 User-Drawn-Sketch-Layer-225) (GlyphFn Object-150 User-Drawn-Sketch-Layer-225))
  - ? A (spatiallyIntersects (GlyphFn Object-150 User-Drawn-Sketch-Layer-225) (GlyphFn Object-147 User-Drawn-Sketch-Layer-225))
  - ? A (spatiallyIntersects (GlyphFn Object-153 User-Drawn-Sketch-Layer-225) (GlyphFn Object-147 User-Drawn-Sketch-Layer-225))

Select the things your bundle represents by typing the name of a collection in the yellow box at the top right. The larger yellow box to the right will show valid completions; use the arrow button to add one of the completions the list of things represented by this bundle.

ShoppingCart:

ShoppingCart is the collection of handcarts that are designed to hold dry goods for shoppers.
Providing Visual/Conceptual Relations

**Bundle Shopping Cart Anatomy:**

Conceptual relationships between **Body** and **Front leg**:

<table>
<thead>
<tr>
<th>User supplied relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which of the following best describes the relationship between Body and Front leg?</td>
</tr>
<tr>
<td>(connectedAtEnd Front leg Body)</td>
</tr>
</tbody>
</table>

Conceptual relationships between **Body** and **Handle**:

<table>
<thead>
<tr>
<th>User supplied relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which of the following best describes the relationship between Body and Handle?</td>
</tr>
<tr>
<td>(connectedAtEnd Handle Body)</td>
</tr>
</tbody>
</table>

Conceptual relationships between **Ground** and **front wheel**:

<table>
<thead>
<tr>
<th>User supplied relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which of the following best describes the relationship between Ground and front wheel?</td>
</tr>
<tr>
<td>(above-Touching front wheel Ground)</td>
</tr>
</tbody>
</table>

Conceptual relationships between **Front axle** and **front wheel**:

<table>
<thead>
<tr>
<th>User supplied relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which of the following best describes the relationship between Front axle and front wheel?</td>
</tr>
<tr>
<td>(alignedCylinderWithin Front axle front wheel)</td>
</tr>
</tbody>
</table>
How visual/conceptual relations are hypothesized

- Qualitative topology used to suggest initial candidates
  - \(\text{insideInSketch } o_1 \ o_2\) if \(\text{glyph } o_1\) is inside \(\text{glyph } o_2\)
  - \(\text{atOrOverlapsInSketch } o_1 \ o_2\) if \(\text{glyph } o_1\) touches or overlaps \(\text{glyph } o_2\)

- Possible specializations filtered by argument type relationships

- You can choose more specialized relationship if desired.

- Not an easy problem
  - Worst case: 150 possibilities for \(\text{insideInSketch}\), 204 for \(\text{atOrOverlapsInSketch}\), with ResearchCyc KB
  - For one corpus of 34 sketches:
    - Mean # questions/sketch = 4
    - Mean # candidates to consider per question = 122
Example: Front Wheel/Axle

Conceptual relationships between Front axle and front wheel:

User supplied relationship
Which of the following best describes the relationship between Front axle and front wheel?

- (localityOfObject Front axle front wheel)
- (mainConstituent Front axle front wheel)
- (objectFoundInLocation Front axle front wheel)
- (objectSides Front axle front wheel)
- (physicalDecompositions Front axle front wheel)
- (physicalParts Front axle front wheel)
- (physicalParts-Separated Front axle front wheel)
- (physical Portions Front axle front wheel)
- (physicallyContains Front axle front wheel)
- (pigments Front axle front wheel)
- (pluggedInto Front axle front wheel)
- (properPhysicalDecompositions Front axle front wheel)
- (properPhysicalParts Front axle front wheel)
- (properSpatiallySubsumes Front axle front wheel)
- (properSpatiallySubsumes-Nontangential Front axle front wheel)
- (properSpatiallySubsumes-Tangential Front axle front wheel)
- (protectiveContains Front axle front wheel)
- (protrudesInto Front axle front wheel)
- (screwedIn Front axle front wheel)
- (spans-Bridgelike Front axle front wheel)
- (spatiallyContains Front axle front wheel)
- (spatiallyIncludes Front axle front wheel)
- (spatiallySubsumes Front axle front wheel)
- (sticksInto Front axle front wheel)
- (sticksInto2D Front axle front wheel)
- (subRegions Front axle front wheel)
- (surfaceParts Front axle front wheel)
- (surrounds-3D Front axle front wheel)
- (surroundsCompletely Front axle front wheel)
Suggesting visual/conceptual relations by analogy

Analogical inferences are surmises, not certainties.

MAC/FAC

Knowledge Base (including case libraries of examples)

Candidate Inference Extraction

Suggestions Filtering

User supplied relationship
Which of the following best describes the relationship between Bin and Chassis?

- (connectedAlongSurface Chassis Bin)
- (rotationallyConnectedTo Chassis Rear Axle)
- (connectedAtEnd Chassis Rear Wheel)
- (above-Touching Front Wheel Ground)

Relationships suggested by analogy

109 candidates
184 candidates
189 candidates
109 candidates